Improving the Agility of BGP Routing

Thomas Wirtgen

Thesis submitted in partial fulfillment of the requirements for the Degree of Doctor in Applied Sciences

October 2023

ICTEAM
Louvain School of Engineering
Université catholique de Louvain
Louvain-la-Neuve
Belgium

Thesis Committee:

Pr. Charles **Pecheur** (Chair)	UCLouvain/ICTEAM, Belgium
Pr. Cristel **Pelsser**	UCLouvain/ICTEAM, Belgium
Pr. Etienne **Rivière** (Secretary)	UCLouvain/ICTEAM, Belgium
Pr. Marco **Chiesa**	KTH, Sweden
Pr. Matthias **Wählisch**	TU Dresden, Germany
Pr. Olivier **Bonaventure** (Advisor)	UCLouvain/ICTEAM, Belgium

Improving the Agility of BGP Routing

by Thomas Wirtgen

ICTEAM
Université catholique de Louvain
Place Sainte-Barbe, 2
1348 Louvain-la-Neuve
Belgium

This work was partially supported by a F.R.S-FNRS FRIA scholarship (Fund for Research training in Industry and Agriculture).

"Sometimes we do get taken by surprise.
For example, when the Internet came
along, we had it as a fifth or sixth priority."

– Bill Gates, 1998

Abstract

The Border Gateway Protocol (BGP) is the key protocol for interconnecting networks on the Internet, enabling its basic functionality. Its origins date back to the early days of the Internet, in 1989.

In the following decades, the Internet has evolved significantly and is now an integral part of economic and social life. However, the design of the BGP protocol no longer meets modern needs. Although efforts to enhance the protocol over time, certain elements of its design are resistant to change, due to its critical role in the Internet infrastructure.

This thesis aims to redesign BGP routing by focusing on three key aspects of modern routing protocols. First, it explores the possibility of allowing network operators to update their routers, freeing them from vendor constraints that inhibit protocol innovation. By introducing the eBPF virtual machine into BGP implementations, network operators gain the ability to design custom features without relying on the IETF or router vendors.

Second, the thesis deals with the modernization of BGP message transport. While the TCP protocol was once sufficient to ensure reliable transmission of routing messages, it now suffers from a number of limitations including its lack of security features. Since 2021, a new transport protocol called QUIC has been standardized by the IETF. QUIC combines new features like connection migration and stream multiplexing with the security features of TLS and the reliability of TCP. The thesis demonstrates the advantages of using QUIC and its characteristics for routing protocols.

The third and final part focuses on improving the security of BGP routing. By default, BGP assumes that advertised routes are usable and accessible in the data plane without verification, which is often a false assumption. To overcome this problem, we propose a system to validate and ensure the reachability of BGP-learned routes in the data plane.

Preamble

The Internet has become an important element of our society, evolving from its origins as an academic research network to a vital component of the global economy and human interaction. In 2023, it is estimated that 66% of the world's population will be using the Internet to access a wide range of resources, with an average number of 3.6 Internet-connected devices per capita [Cis20].

With over 80,000 Autonomous Systems (ASes) facilitating connectivity [Hus23b], the Internet continues to expand, enabling more people to be connected and more services to emerge. To communicate with each other, network equipment suppliers such as Cisco, Nokia and Juniper offer routers that require manual configuration. Correctly configured, these routers exchange routing information to enable communication between networks, enabling them to share internal policies and configure parameters according to specific needs.

To ensure interoperability between routers from different vendors, standardized routing protocols such as BGP (Border Gateway Protocol), OSPF (Open Shortest Path First), IS-IS (Intermediate System to Intermediate System) or EIGRP (Enhanced Interior Gateway Routing Protocol) are used. These protocols, standardized by organizations such as the Internet Engineering Task Force (IETF), form the basis for cooperation between routers. However, these protocols were established, in the late 1980s, before the exponential growth of the Internet. The evolving requirements of modern routers do not match their initial design. As a result, the evolution of distributed routing is slow for historical reasons, and it takes a long time to introduce new features.

Although suggestions, improvements and additions to these protocols are proposed every year, some intrinsic aspects of the design cannot be easily corrected. Routers of the 1980s did not have the computing power to integrate advanced technologies for disseminating routing information. As a result, compromises were made, and this ossification became the norm for routers willing to participate to the Internet.

The objective of this thesis is to reassess the architecture of distributed routing protocols and update them to align with current technologies and requirements. More specifically, this thesis focuses on BGP, which is the key routing protocol on the Internet. BGP facilitates the exchange of routing information between Internet sub-networks and also plays a crucial role in the dissemination of this information within each sub-network, which gives it considerable importance in the Internet ecosystem.

The contributions of this thesis are outlined as follows:

- *Bringing routing protocol implementation extensibility with plugins.*

  Routing protocol implementations are generally managed by router vendors. They are mainly closed source and therefore do not offer network operators the possibility to evolve them. This constraint strongly limits the operators on what they can deploy in their network whose needs are constantly evolving. In addition, the introduction of a new functionality in a protocol requires long steps that slow down innovation. For all implementations to be interoperable, the functionality must first be standardized by the Internet Engineering Task Force (IETF), which often takes time, and then vendors must implement, test and deploy it on their operating systems.

  We propose a new approach that allows network operators to extend a specific implementation of a routing protocol themselves through plugins. The plugins can extend the protocol and modify the protocol's internal routines through a simple API. We modified the BGP and OSPF implementations of FRRouting, an open-source implementation of several routing protocols, to demonstrate the applicability of such an approach.

- *Designing a routing protocol to be extensible, regardless of the underlying implementation.*

  We have shown that it is possible to augment an implementation of a routing protocol with plugins that network operators develop and inject into their routers. However, this approach has a major limitation. An operator typically has various routers from different vendors with different operating systems. This means that the underlying implementation of a routing protocol is also different. A plugin developed for one implementation will not work in another implementation of the same routing protocol. Nevertheless, all implementations must at least respect the routing protocol standard. It is therefore possible to abstract the core data structures and routines that each implementation must maintain to comply with the standard

  We propose an approach to abstract a routing implementation and thus allow a once-written plugin to be executed on any implementation of a routing protocol. To demonstrate the feasibility of this approach, we prototype *x*BGP, a vendor-neutral API that exposes the key data structures and functions of any BGP implementation. To show that the multi-vendor approach is possible, we adapt two open-source BGP implementations to make them *x*BGP compatible and demonstrate several use cases that work on both implementations.

- *Modernizing the transport of routing messages.*

  BGP uses plain TCP connections to exchange routing messages. However, TCP does not have strong security features. To address this, the IETF and operators have standardized authentication methods like TCP-MD5 or TCP-AO, and other techniques to protect against various types of attacks. Recently, the QUIC protocol has been standardized, which has shown that a transport protocol can be both secure and efficient. QUIC is being rapidly adopted by the use of HTTP/3 and DNS over QUIC.

  We are looking into how a secure transport protocol like QUIC can leverage routing protocols. By using QUIC, we not only secure the exchange of routing information but also make BGP more flexible. We have added QUIC to the open-source BIRD routing daemon to demonstrate this flexibility. We show that using BGP over QUIC is more flexible than using plain TCP. With BGP over QUIC, we can establish BGP connections as needed without compromising the security of routing information. BGP over QUIC also improves remote blackholing services and makes it easier to support complex BGP filters.

- *Securing and making the protocol aware of its environment.*

  BGP is a fragile routing protocol since it is based on an implicit system of trust between the Autonomous Systems (AS) participating in the exchange of routes on the Internet. Any router can announce the routes it wants without being the owner. Due to the lack of a validation system for the announcements made by BGP routers, a series of RFCs published after the release of BGP have partially solved this problem by introducing the Resource Public Key Infrastructure (RPKI).

  We aim to complement the security mechanisms of BGP by introducing a new active control system. We propose to validate BGP paths in the dataplane. We extend the BGP implementation of FRRouting (an open source Internet routing protocol suite) to demonstrate the feasibility of our approach. Finally, we discuss the potential of an active system in a routing protocol to both secure BGP announcements and improve the routing decision. We believe that augmenting a routing protocol by using data plane metrics can lead to better route propagation.

The thesis is structured in five parts:

- **Part I** introduces the key concepts of routing in the context of the Internet (**Chapter 1 (Introduction)**).
- **Part II** tackles the programability of routing protocols and proposes a technique to enable a network operator to program their routers. It contains the following chapters:
  - **Chapter 2 (Motivations)** motivates the need for programability in routing protocols.
  - **Chapter 3 (Augmenting BGP with Plugins)** discusses the extension of a routing protocol implementation using plugins.
  - **Chapter 4 (*x*BGP)** explores the concept of using the same plugin on different implementations of a same routing protocol.
- **Part III** proposes to explore the new possibilities offered to routing protocols by using a secure transport protocol to exchange routing messages. In particular:
  - **Chapter 5 (Replacing TCP with QUIC)** proposes the replacement of TCP, the transport protocol used by BGP, with QUIC.
- **Part IV** focuses on the aspect of combining control-plane and data-plane information to provide a better routing service. The first step is performed with:
  - **Chapter 6 (Securing BGP routes in the data-plane)** which examines an approach to secure BGP routes through secure handshake establishments in the data plane.
- **Part V** concludes this thesis with:
  - **Chapter 7 (Future work)** which addresses some of the future research questions raised throughout this thesis.
  - **Chapter 8 (Conclusion)** which concludes this thesis.

Bibliographic notes

This thesis led to the publication of, and is based on, the following works:

Conference Publications

1. T. Wirtgen, C. Dénos, Q. De Coninck, M. Jadin, and O. Bonaventure. "The Case for Pluginized Routing Protocols". In: *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE. 2019, pp. 1–12. DOI: 10.1109/ICNP.2019.8888065.

2. T. Wirtgen, T. Rousseaux, Q. De Coninck, N. Rybowski, R. Bush, L. Vanbever, A. Legay, and O. Bonaventure. "xBGP: Faster Innovation in Routing Protocols". In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 35–50. ISBN: 978-1-939133-27-4. URL: https://www.usenix.org/conference/nsdi23/presentation/wirtgen.

Workshop Publications

1. T. Wirtgen, Q. De Coninck, R. Bush, L. Vanbever, and O. Bonaventure. "xBGP: When You Can't Wait for the IETF and Vendors". In: *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. HotNets '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1–7. ISBN: 9781450381451. DOI: 10.1145/3422604.3425952. URL: https://doi.org/10.1145/3422604.3425952.

    - Received the 2021 IETF/IRTF Applied Networking Research Prize (ANRP).

2. T. Wirtgen and O. Bonaventure. "A First Step towards Checking BGP Routes in the Dataplane". In: *Proceedings of the ACM SIGCOMM Workshop on Future of Internet Routing & Addressing*. FIRA '22. Amsterdam, Netherlands: Association for Computing Machinery, 2022, pp. 50–57. ISBN: 9781450393287. DOI: 10.1145/3527974.3545723. URL: https://doi.org/10.1145/3527974.3545723.

Posters and Demos

1. T. Wirtgen, N. Rybowski, C. Pelsser, and O. Bonaventure. "Routing over QUIC: Bringing transport innovations to routing protocols". In: *Poster Session of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, 2023.

Acknowledgments

The completion of this thesis would not have been possible without the help of several people whom I would like to thank.

First and foremost, my academic and professional career has been largely guided by the unfailing support and advice of Professor Olivier Bonaventure, my advisor. His unshakeable faith in my abilities and his constant advice transformed my aspirations into concrete achievements. I credit him for allowing me to carry out all of the contributions presented in this thesis.

I am also deeply grateful to Quentin De Coninck for his exceptional mentorship. He guided me not only through my master's thesis, but also through most of my doctoral training. Our in-depth discussions and the scrutiny of my results have greatly enhanced my competences as a researcher.

I would like to express my gratitude to the members of my thesis committee: Matthias Wählisch, Marco Chiesa, Cristel Pelsser, Etienne Rivière and Charles Pecheur. Their helpful comments during my private defense greatly enhanced the quality of my thesis. I am grateful for their participation in my committee and for the time they spent reading and reviewing my work in detail.

This thesis is also the fruit of collaborations with other people whom I would like to thank: Quentin De Coninck, Mathieu Jadin, Cyril Dénos, Nicolas Rybowski, Tom Rousseaux, Randy Bush, Laurent Vanbever, Cristel Pelsser and Axel Legay.

I am grateful to my former IP Networking Lab colleagues – Viet-Hoang Tran, Fabien Duchêne, Mathieu Jadin, Florentin Rochet, Tom Barbette, Quentin De Coninck, Maxime Piraux, François Michel, Louis Navarre and Nicolas Rybowski. Our many discussions and interactions indirectly played an important role in the elaboration of the ideas and concepts presented in this thesis.

I would like to express my deep gratitude to INGI's administrative and technical staff, and in particular to Vanessa Maons for her invaluable help with all administrative and logistical matters, to Sophie Renard for her invaluable help with accounting, and to Nicolas Detienne and Anthony Gégo for their exemplary responsiveness to my sometimes far-fetched technical requests.

Finally, over the last few years, I have been fortunate to enjoy the unfailing support and encouragement of my family and friends. Although they may not have understood all the subtlety of my work, their support has been the cornerstone that has enabled me to complete this thesis.

Thomas Wirtgen
October 4, 2023

Contents

Part I

Background

Internet Routing

1

In this chapter, we discuss the network concepts and protocols necessary to understand this thesis. Specifically, this chapter explains how to transmit data between the nodes that are part of the Internet. We begin by introducing the high-level architecture of the Internet in Section 1.1. Next, Section 1.2 explains the basics of routing, the process that enables to deliver data to the right Internet destination. We continue in Section 1.3 by explaining the operation of a router, which is the main component of the Internet. We then explain in Section 1.4 that routers exchange their routing knowledge using a routing protocol. The Border Gateway Protocol (BGP), which is the focus of most of this thesis, will be explained in Section 1.5. BGP was originally designed without much emphasis on security. Since its standardization, additional security extensions have been proposed. In Section 1.6, we discuss the main security measures implemented to protect routing information from manipulation or misconfiguration. Finally, in addition to BGP, which is a decentralized protocol, there are also centralized approaches to routing. Section 1.7 discusses the main centralized solutions that are currently in use.

1.1 The High-Level Organization of the Internet

In today's networks, devices exchange data with each other by using a common infrastructure. The most basic form of network infrastructure is the Local Area Network (LAN). It allows multiple devices to connect and communicate with each other within a small physical area, such as a home or office. In a LAN, devices can communicate directly with each other through a cable or wireless connection, without needing to go through an intermediate device. Numerous local networks can be created within the same company. For example, one can have a dedicated LAN for each type of service, thus separating devices of different natures. To enable communication between different LANs, an intermediate device called a switch is used. However, LANs are limited in scope and cannot provide connectivity between networks that are physically separated. A router is then required to enable communication between different LANs.

A router forwards data between networks. It uses routing protocols to determine the best path for the data to take through the network. Routers

are an essential part of modern networks and use algorithms, called routing protocols, to communicate information about the network topology to other routers. They use this information to build a routing table, which contains information about the best path to take to reach a particular destination.

There are three main types of routing protocol: distance-vector, link-state and path-vector. Each type has its own advantages and limitations, and is adapted to particular networks and environments. For example, distance-vector protocols are simpler and less CPU-intensive. This makes them suitable for smaller networks. In contrast, link-state protocols are more complex, but provide more precise information about the network topology.

Routing protocols are fundamental to modern networks. They enable communication between devices on different networks. Choosing the right protocol can have a significant impact on network performance and reliability. Hence, network administrators and engineers need to be aware of the strengths and weaknesses of different protocols.

Routing protocols also play an important role in enabling the Internet to function as a global network. Internet Service Providers (ISPs) interconnect their networks using routers, creating a sequence of connected networks spanning the globe. When a device on one network wants to communicate with a device on another network, data must be routed through multiple routers along the path to its destination. Routing protocols are responsible for determining the best path to take through the network, based on factors such as link capacity, congestion levels, and network topology.

There are different types of provider: some operate globally, others nationally or regionally, and still others locally, with limited access to the Internet. In general, local providers rely on other ISPs, or their own provider, to enable their end users to connect to the network.

In some cases, ISPs choose not to establish direct links with each other due to geographic constraints. In these cases, they rely on Internet Exchange Points (IXPs) to facilitate peering connections. IXPs are physical locations where various network operators, including ISPs, can exchange data directly with each other. These points are designed to provide a more efficient and direct method of exchanging traffic, eliminating the need to route all data through a third-party network. By relying on IXPs, networks can establish peering links and exchange data directly, without the need for direct physical connections. Without IXPs, if two networks are not directly connected, traffic would have to traverse intermediate networks, which may imply additional latency. By connecting to an IXP, ISPs can thus reduce the cost of exchanging traffic and improve the performance of their networks by reducing the number of network hops required to reach their destination. IXPs also increase network resiliency, as they provide an alternative path for traffic if one of the ISPs' links fails. IXPs are typically located in major cities around the world and can

range in size from small regional exchanges to large international hubs. They are managed by neutral organizations that ensure fair and open access to all participating networks.

1.2 IP Routing

The Internet depends on network infrastructure and routing protocols to function, but that is not enough to establish full connectivity between two hosts. Today, the Internet is built upon the Internet Protocol (IP) [Inf81; HD98]. Every device willing to connect to the Internet possesses a unique address of either 32 bits (IPv4 [Inf81]) or 128 bits (IPv6 [HD98]), enabling it to send and receive packets.

IPv4 and IPv6 addresses are represented in a way that is easier for humans to remember, rather than writing them in binary format. An IPv4 address is represented by four 4-byte decimal numbers ranging from `0` to `255`. For example, the IPv4 address `192.0.2.42` is actually treated by a network node as the binary address `11000000 00000000 00000010 00101010`. By contrast, an IPv6 address is longer than an IPv4 address and follows a different notation. It consists of eight 16-bit fields separated by colons. Each field represents a valid hexadecimal number between `0` and `ffff`. For example, an IPv6 address may look like this:

```
2001:0db8:0000:0000:0000:0000:0000:c0de
```

This representation is written in an "exploded" notation. This means that all fields of the IPv6 address are displayed. However, To simplify and shorten IPv6 addresses, a shorthand notation `"::"` is used. This notation removes consecutive null hexadecimal fields (`0` or `0000`), reducing the address's textual length and saving space. Additionally, leading zeroes at the beginning of a 16-bit field can be omitted since they do not provide additional information. By applying these optimizations, the previous IPv6 address can be rewritten as `2001:db8::c0de`.

Note that the shorthand notation "`::`" in an IPv6 address must only be used once. For example, in the address `2001:db8::cafe::c0de`, we cannot determine the exact number of null fields between the two "`::`" notations, as this is ambiguous. Therefore, `2001:db8::cafe::c0de` is not a valid IPv6 address.

To transmit data, the sending device breaks it down into smaller units known as packets. Each packet is marked with a source IP address, indicating the end host that initiated it, as well as a destination IP address representing the intended recipient. Since end users are typically not directly linked, packets are forwarded through intermediate routers. These routers pass the packets along to the next router in the path until they ultimately reach their intended

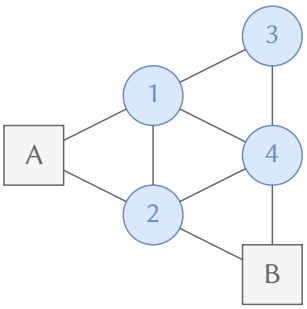


Figure 1.1: Example of a simple network topology.

destination. This process of sequentially routing packets from one router to another is known as IP forwarding.

When a router receives a packet, it checks the destination IP address to find out where to send it next. This may be another router or the final destination. To do this, the router must know the next node or "next hop" to contact. This process of finding the appropriate next hop is known as routing and is usually done using a routing protocol.

One way to establish routes in a network is to manually configure each router with static routes to each destination. However, this approach has limitations. First, it lacks scalability, as networks often consist of thousands of devices. Configuring routes individually on each router is error-prone and unmanageable on a human scale. Second, static routing does not automatically handle link failures. If a link between two routers experiences a problem, packets will continue to be routed through that link since the route remains configured. This is why dynamic methods are prefered. They use dynamic routing protocols that quickly detect failures in order to reroute traffic over other valid and usable links. These protocols automatically discover the paths to each destination, using algorithms that aim to select the available shortest path.

Consider the network topology described in Figure 1.1. The routers are represented by a number in a blue circle and the end hosts by a letter in a gray square. To establish a connection between A and B, several paths are available. For example, a valid path is as follows: A $\longrightarrow$ 1 $\longrightarrow$ 3 $\longrightarrow$ 4 $\longrightarrow$ B. This path is not optimal, there is a shorter path that only passes through node 2. Algorithms such as Dijkstra [Dij59] or Bellman-Ford [Bel58] can be used to determine the ideal path, creating the shortest path from any router to each destination. The notion of the shortest path is specific to each ISP, some determine the path according to their routing policies, others according to the capacity of the links between the routers, or according to the network load.

Typically, ISPs use a shortest path algorithm to calculate routes based on

the metric they wish to optimize. However, when it comes to inter-ISP routing on the Internet, other methods are needed to learn the paths due to the sheer size of the Internet.

As of this writing, there are over 80,000 interconnected ISPs [Hus23b]. Even with a conservative estimate of 10 routers per ISP (which is unrealistic because ISPs often have a larger number of routers [MMD22; Spr+04]), the Dijkstra and Bellman-Ford algorithms would have to handle over 800,000 nodes. This poses significant computational and memory management challenges. In addition, each ISP defines its own routing policies, which can lead to conflicting routing decisions and difficulties in reaching agreements between ISPs.

For example, one operator may prioritize minimizing the geographic distance traveled within its network, while another operator may focus on minimizing the load on its links. This may result in fewer optimal paths from the other operator's perspective. Such discrepancies in routing objectives can make the routing agreement between different ISPs more complex and difficult to resolve.

To address these challenges, the entire ISP network is treated as a unified entity. Each ISP knows how to reach all other ISPs, but does not have detailed information about the internal architecture and routing policies of individual ISPs. Therefore, to enable communication between different ISPs, inter-ISP routing algorithms consider each destination within an ISP as part of a larger group referred to as a "prefix", which greatly reduces the computation and amount of state stored per router when computing the shortest path for each ISP.

Figure 1.2 provides an overview of the routing structure found on today's Internet. Each ISP consists of interconnected routers connected to one or more LANs. In our simplified model, we assume that the end hosts connected to a router belong to the same LAN, as shown in Router 8 of ISP 2. Within each ISP, one or several Interior Gateway Protocols (IGPs) are used, tailored to the specific routing policies of that ISP. This allows for seamless communication between all routers and end hosts within the ISP.

To facilitate communication between ISPs, a specialized routing protocol called Border Gateway Protocol (BGP) [RHL06] is used. BGP treats each ISP as a collective entity or aggregate, which simplifies the process of discovering and establishing connections with all other ISPs on the Internet. Unlike the routing protocols used for internal communication within an ISP, BGP is specifically optimized to handle the current scale of the Internet. More details on BGP will be provided in the 1.5 section.

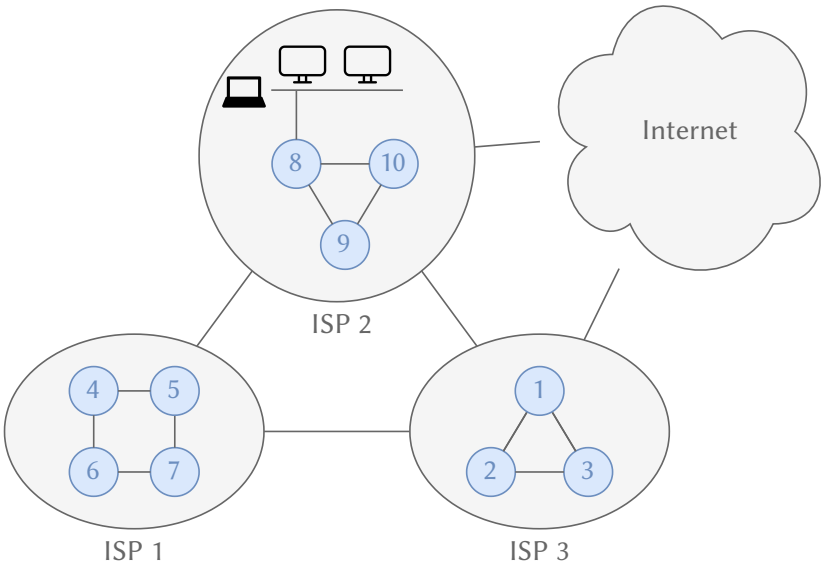


Figure 1.2: Example of a simplified model of the Internet.

IP Prefixes

A routing protocol is responsible for discovering the paths to each IP address within a network. In the case of IPv6, if we consider addresses as indistinguishable entities, there could be about 2^{128} (approximately 10^{38}) different routes. However, it is practically impossible for routers to manage and maintain such many routes. To solve this problem, routers use route aggregation by grouping IP addresses with the same prefix into a single routing entry.

For example, consider the IPv6 prefix `aa:bbbb:cccc::/48`, which encompasses all IPv6 addresses sharing the first 48 most significant bits (i.e., $2^{80} \approx 10^{24}$ IPv6 addresses). Therefore, `aa:bbbb:cccc::1` and `aa:bbbb:cccc::ffff` both belong to the same IP prefix. These prefixes are used as the basis for routing information exchanged between routers. By using IP prefixes, inter-ISP routing becomes more scalable and reduces the amount of state maintained per router.

ISPs obtain specific IP prefixes, from organizations supervised by the Internet Assigned Numbers Authority (IANA). These IP prefixes are distributed to ISPs via regional Internet registries (RIRs) such as RIPE NCC or APNIC. Each ISP can receive one or several IP prefixes from these RIRs. For example, in Figure 1.2, ISP 1 can receive `198.51.100.224/27`, ISP 2 receives `198.51.100.160/27` and ISP 3 receives `198.51.100.32/27`. Within their respective networks, ISPs allocate these prefixes to their individual nodes

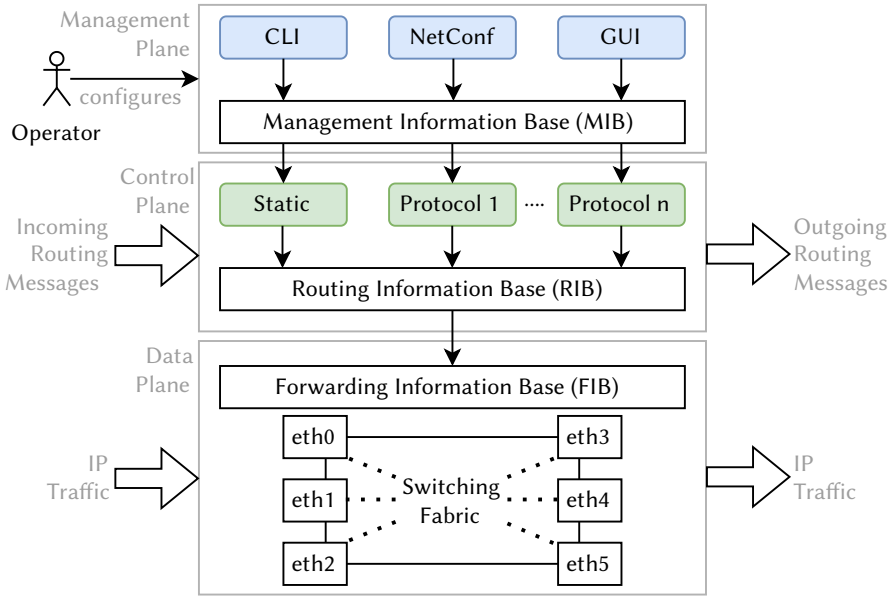


Figure 1.3: High-level architecture of a traditional IP router. It can be modeled in three layers: management, control and data.

according to their own needs. Routing protocols use these prefixes to determine the appropriate path to forward data to a specific ISP on the Internet. When a packet originating from the Internet has the destination address of `198.51.100.225`, which is within the IP prefix assigned to ISP 1, the routing protocol can choose between two paths to forward this packet. The first path is via ISP 3, and the second path is via ISP 2.

1.3 IP Routers

A traditional IP router can be represented as in Figure 1.3. The architecture of such a router can be divided into three main layers.

The management-plane. The management plane allows network operators to configure and control the behavior of their routers. Using the router's built-in operating system, operators can fine-tune various parameters, enable specific routing protocols, add static routes, or set up filters to regulate the distribution of IP prefixes on the network.

The command line interface (CLI) is a commonly used method to interact with routers. It provides operators with a simple syntax for communicating and configuring desired parameters. The CLI is accessible via a console port directly connected to the router or via protocols such as Telnet [PR83] or

SSH [LY06] when remote access is required.

However, for large networks, manually configuring each router becomes a tedious and error-prone task. To simplify this process, router vendors often include additional interfaces to automate the configuration procedures. This is the case with the Network Configuration Protocol (NetConf) [Enn+11], which allows devices to be configured using the YANG data model. With NetConf, configuration installations, changes and deletions can be automated, simplifying router management.

Network operators also need continuous monitoring and analysis of their network and equipment. Routers have built-in monitoring solutions that provide valuable information about network performance. These solutions offer a variety of statistics, including CPU and memory usage, network protocol events, interface status, number of packet transfers, and more. Operators can retrieve this information using protocols such as SNMP (Simple Network Management Protocol) [Fed+90] or IPFIX (Internet Protocol Flow Information Export) [ACT13] to effectively monitor and analyze the network and its devices.

The control-plane. Once the router is configured, the control plane comes into action, retrieves the router's configuration and begins to discover the paths to each IP prefix. The protocols enabled on this plane are used to populate the Routing Information Base (RIB), which contains the routes the router should follow to reach each prefix. The operator has the option of manually adding static routes to the RIB using the STATIC routing process. While this manual configuration of the routing table has its limitations, it can be useful in certain situations. For example, in small networks with a single exit path, it may be more advantageous to set up a static route rather than relying on a dynamic routing protocol.

Various dynamic routing protocols can be enabled on the control plane, including OSPF [Moy98], BGP [RHL06], RIP [Mal98], EIGRP [Sav+16] or IS-IS [ISO]. OSPF allows for the exchange of routes within the same network, such as an ISP or a cloud service provider. Other protocols such as RIP, EIGRP and IS-IS, also known as Interior Gateway Information (IGP) protocols, are used to exchange information within the same network. The choice of protocol depends on the requirements of the operator and the size of the network. Each protocol has its own way of exchanging routing information.

OSPF and IS-IS are classified as link-state protocols, in which routers exchange adjacency information to establish a connectivity map. Among other things, this information indicates which router interface is connected to another router interface. In contrast, RIP is a distance vector protocol, in which routers calculate the distance towards an IP prefix based on the distance shared by the router that provided the information.

There are also hybrid protocols such as EIGRP, which combine elements of both distance vector and link state protocols in their operation.

To exchange routes between different domains or subnets, the Border Gateway Protocol (BGP) is widely used on the Internet. As of February 2023, BGP facilitates the exchange of over 1.2 million IPv4 and IPv6 prefixes [Hus23a]. Because of the large number of routes it manages, BGP is neither a distance vector nor a link state protocol. Simply announcing a distance, as in distance vector protocols, would cause convergence problems. Flooding routers with detailed adjacency information, as in link-state protocols, would be impractical in terms of memory usage. Instead, BGP is a path vector protocol. It announces the path (i.e., the ordered sequence of subnets to follow) to a prefix, which the router uses to determine the best route to the destination IP prefix.

The data-plane. The data-plane is probably the most important component of an IP router because it is responsible for forwarding IP packets. It uses the information provided by the routing protocols in the RIB to calculate the Forwarding Information Base (FIB). The FIB is designed to be implemented at the hardware level, enabling fast lookup of information. In general, each FIB entry represents the best route for a specific IP prefix and includes the next hop through which the packet should be routed. However, in some situations, the routing protocol of the control-plane identifies several best routes having the same cost. In this case, the router adds all these routes to the FIB and forwards packets to the destination using these different paths. This technique is called Equal-Cost Multi-Path (ECMP) routing. To determine which path to choose for packet forwarding, a simple method would be to alternate between the available paths, ensuring an equal distribution of packets over all paths. However, this approach is not suitable for reliable transport protocol such as TCP. In fact, when several paths are used for the same TCP connection, packets belonging to a single TCP flow may be rearranged, potentially causing packets to arrive at their destination out-of-order. This problem can have various consequences. First, it can lead to an increase in latency, as TCP has to wait for out-of-order packets to be reassembled before transmitting them to the application. Second, it can lead to unnecessary packet retransmissions. Missing packets can be misinterpreted by the receiver as a packet lost, triggering retransmissions. Finally, TCP's congestion control mechanisms can be disrupted. Some congestion control algorithms rely on packet loss as a signal to adjust the transmission rate of TCP segments. Packet reordering can lead to indications of spurious packet loss. To solve this problem, the router uses a hash function to select the path the packets should take. This hash function will be used to compute a hash value based on various information contained in the packet such as the source and destination IP addresses, source and destination port numbers, and the protocol used. The

formula is then:

$$\texttt{hash(IP}_{\texttt{src}}\texttt{, IP}_{\texttt{dst}}\texttt{, Port}_{\texttt{src}}\texttt{, Port}_{\texttt{dst}}\texttt{, Protocol) mod N}$$

Where `N` represents the number of equal-cost paths. The efficient calculation of this hash function has been the subject of research in the literature [Hop00]. By applying this formula, all packets belonging to a specific flow will systematically take the same path, effectively mitigating the above-mentioned problems.

In some cases, the FIB may contain overlapping IP prefixes. For example, consider the IPv4 prefixes (*a*) `192.0.2.0/24` and (*b*) `192.0.2.128/25`. If an IP packet has a destination address of `192.0.2.250`, both prefixes contain forwarding information. To resolve this ambiguity, routers use a longest prefix matching policy. In the given example, prefix (*b*) `192.0.2.128/25` would take precedence over prefix (*a*) `192.0.2.0/24` due to its longer matching length.

In addition, the data plane incorporates mechanisms for implementing queuing policies to prioritize certain packets over others. For example, packets carrying real-time video or voice data require low latency and therefore receive higher priority than packets involved in long file downloads. This prioritization of packets is a basic example of quality of service (QoS) mechanisms.

Finally, the data plane supports filtering capabilities based on access control lists (ACLs) defined in the management plane. ACLs specify rules that determine whether a router should forward or block packets based on their addresses or ports. By applying ACLs, routers can implement security policies and control traffic flow.

1.4 The Transport of Routing Messages

The Internet infrastructure and routers allow the exchange of data between hosts, but they do not guarantee a perfect reception of the data. Various factors can cause packets to be corrupted, duplicated, lost or misdelivered. Electromagnetic interference, limited router processing capacity, intentional packet discarding, and rerouting over multiple paths can all contribute to these problems. To ensure reliable transmission of data between hosts, a reliable transport protocol is required.

Transport protocols manage the transmission of data and ensure reliable, lossless and in-order delivery. The Transmission Control Protocol (TCP) [Pos81] is a widely used transport protocol that provides this reliability. It ensures that data sent from the source is received correctly by the destination, even in the presence of network imperfections. Other reliable transport protocols include the Stream Control Transmission Protocol (SCTP) [Ste07] or the QUIC protocol [IT21].

Routing protocols, which use the same infrastructure as the end hosts, are also affected by network imperfections. Even when two routers are directly connected, there is always a risk of packet corruption or loss during transit, especially when the routers span several continents. Therefore, routing protocols also need a reliable transport mechanism to exchange routing information. However, many routing protocols do not use TCP for historical and resource limitation reasons.

Protocols such as IS-IS and OSPF use their own reliable transport protocol implemented on top of the network protocol. These protocols handle imperfections in the network infrastructure and provide a reliable exchange of routing information. The use of custom transport protocols was necessary in the past when routers had limited computing power and memory, making it expensive to maintain TCP sessions.

In the case of RIP, which has been standardized for IP networks, the protocol uses the User Datagram Protocol (UDP) [Pos80] instead of TCP. UDP does not allow for a reliable and orderly data exchange like TCP. However, for sending small amounts of data, UDP is suitable because it imposes less overhead in terms of CPU and memory costs for routers. RIP was also standardized when routers had stronger resource constraints than today, making UDP an appropriate choice as a transport protocol for routing messages.

In contrast, the BGP protocol, relies primarily on TCP as the default choice for reliable transport. TCP was widely available on routers at the time of BGP's initial implementation, making it an appropriate choice. Although the original RFC1105 [LR89] for BGP states that "any reliable transport protocol may be used", TCP became the de facto standard and no extensions have been standardized to use a different transport protocol than TCP since then.

1.5 Inter-domain routing with the Border Gateway Protocol (BGP)

In the context of the Internet, the Border Gateway Protocol (BGP) [RHL06] is the primary protocol used to exchange routing information between networks, called Autonomous Systems (AS). An AS represents a collection of devices in specific IP prefix ranges, governed by one or more entities with shared routing policies. Examples of ASes include French cloud providers like OVH and network operators like Belnet, which connects Belgian universities and some public services.

The main purpose of BGP is to provide connectivity between ASes on the Internet. BGP works as a path vector protocol, where edge routers within ASes exchange information about the paths they know to specific IP prefixes. This information is transmitted in the form of an AS path, which is an ordered list of ASes that IP packets must traverse to reach the destination IP prefix.

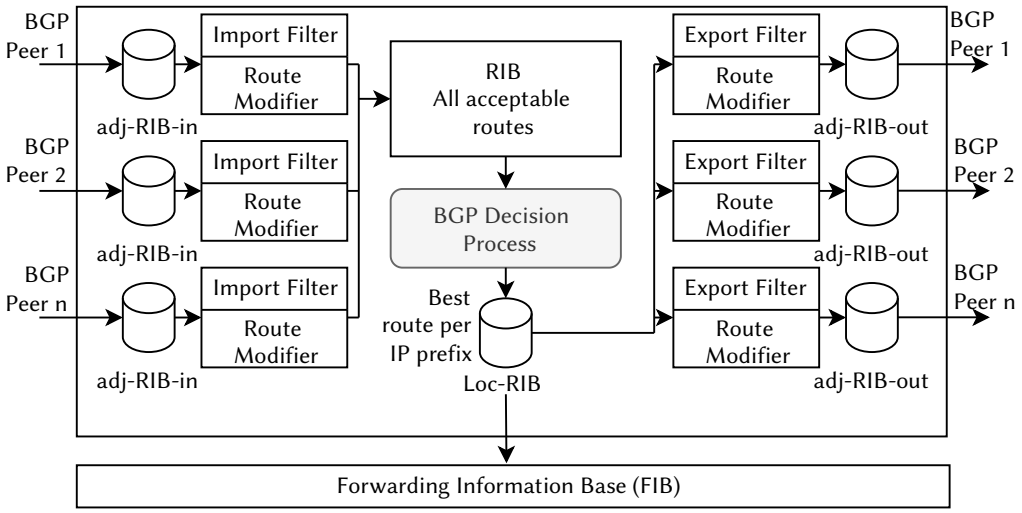


Figure 1.4: High-Level Architecture of a BGP Router.

In addition to the AS path, BGP allows for the exchange of attributes that provide more detailed information to determine the best route to each IP prefix based on the routing policies of each AS. Routers participating in the BGP protocol can learn multiple routes to a particular IP prefix. However, each BGP router selects and shares only the best route with its BGP neighbors.

By exchanging routing information and AS-Paths, BGP enables interconnectivity between ASes, which allows IP packets to be routed over the Internet.

The Organization of a BGP router

The architecture of a BGP implementation in an IP router, as described in RFC4271 and depicted in Figure 1.4, includes several steps in route processing. These steps can be summarized as follows:

In the **first step**, each router sends its best route for each IP prefix in a BGP Update message to its BGP neighbors. These routes are received and stored in the routing table "Adj-RIB-in" (Adjacent Routing Information Base-In). The Adj-RIB-in contains all the unmodified routes received from BGP neighbors.

The routes in the Adj-RIB-in are then subject to a series of filters defined by the network operator. These filters are used to enforce routing policies and ensure that only acceptable routes are considered. During this step, routes can be modified, for example, by assigning them a higher preference or by manipulating their attributes.

After passing through the filters, the acceptable routes are stored in the routing table, known as the Routing Information Base (RIB). The RIB contains

all the routes that the local BGP router has determined to be acceptable based on the routing policies and modifications applied.

In a **second step**, the BGP router initiates the BGP decision process to select the best route for each IP prefix. This process involves comparing route attributes, such as AS-Path length, origin and other configurable BGP attributes. After this step, only the best route for each IP prefix is selected and stored in the Loc-RIB (Local Routing Information Base).

Finally, in the **third step**, the BGP router announces the best selected route for each known prefix to its neighboring routers. Before being advertised, the routes can be subject to another set of export filters, which can modify the route attributes. These export filters can be used for various purposes, such as traffic engineering. Routes that are modified and granted by the export filters are stored in the "Adj-RIB-Out" (Adjacent Routing Information Base-Out) and are eventually sent to other BGP routers on the network.

Not all routers keep separate Adj-RIB-in or Adj-RIB-out tables because of performance or memory limitations, but these tables are helpful for specific enhancements and functionalities like graceful restart [Rek+07]. Also, to optimize memory usage, routes are not duplicated in all tables. Instead, only one copy of the route is kept and each table references it with its own data, including changes made by import or export filters.

The Type of BGP Sessions

BGP can be used in two ways: **external** BGP (eBGP) and **internal** BGP (iBGP). First: eBGP is used to announce routes between different autonomous systems (ASes). ASes use eBGP sessions to exchange routing information with neighboring ASes. These eBGP sessions are typically established between routers located at the edge of the ASes. Routes learned from eBGP sessions are stored in routing tables and can be further processed and selected as the best routes.

Second: iBGP is used within an AS to propagate external routes learned by eBGP to all routers in the AS. For a full-routing table such as those found on the Internet, it is not possible to re-announce them in a link-state or distance vector protocol. These protocols simply could not handle the load. The BGP standard specifies that routers must not redistribute routing information received from one internal peer to another internal peer. Therefore, all routers willing to participate in the iBGP network should be connected in a full-mesh topology. In a full-mesh iBGP configuration, each router in the AS establishes a direct iBGP session with all other routers in the AS. This ensures that all routers in the AS have consistent routing information. However, configuring and maintaining a full-mesh iBGP can be impractical and difficult to scale as the number of routers increases. For a network with n routers, the operator must

maintain $\frac{n(n-1)}{2}$ iBGP sessions. In practice, the operator must configure all routers; hence it must apply $(n-1)$ iBGP configurations per router, therefore $n(n-1)$ iBGP configuration to build the iBGP full mesh.

To address scalability issues, the concept of route reflection (RR) was introduced [CBC06]. In a route reflection configuration, one or more routers act as route reflectors. These route reflectors receive routes from their iBGP peers and reflect these routes to other iBGP routers, called clients. In the iBGP full mesh, routes learned over one iBGP session should not be reanounced on another iBGP session to avoid control-plane loops. However, a route reflector can readvertise iBGP routes to another iBGP client. Hence, to avoid loops in the distribution of routing information within the iBGP configuration, two new attributes are introduced:

- Originator-Id: This attribute is added by the first route reflector that reflects a route. The Originator-Id is set as the router ID of the reflector. When a router receives a route with its own Originator-Id, it knows that the route has already been reflected in its own AS and can discard it to avoid routing loops.
- Cluster-List: This attribute is a list of route reflectors that have reflected the route. This attribute is used to identify loops in the iBGP configuration. If a route reflector sees its own Cluster-Id in the list, it knows that the route has already passed through it and can be dropped to avoid loops.

By using the Originator-Id and Cluster-List attributes, iBGP implementations can ensure loop-free distribution of routing information while reducing the number of iBGP sessions required in the network.

The Type of BGP Messages

The BGP protocol uses different types of messages to communicate information about routing and informative events to other BGP routers.

- **Open**. This message is sent to establish the BGP connection after the TCP connection has been opened. It contains basic information about the router, such as BGP version, AS number, router ID and hold timer. It can also contain optional parameters for negotiating certain BGP extensions, such as graceful restart [Rek+07], route refresh [Che00] or extended BGP messages [BPW19].
- **Keepalive**. This message is sent periodically by the BGP router once the BGP session has been opened. Its main purpose is to maintain the

BGP connection up. If the router does not receive a BGP Keepalive message within the period agreed when the connection was established with the hold timer, the session is considered lost.

- **Update**. This is the most important message, as it contains information about the routes reachable by the BGP router. It can also be used to delete routes when a router can no longer reach them, or to modify routes that have already been advertised.
- **Notification**. This message is sent before a BGP session is closed. It contains information enabling the remote router to understand why the BGP connection is closed. Possible reasons are errors in the BGP connection, such as a malformed BGP message, insufficient resources or an administrative shutdown initiated by the network operator.

In the initial design of the BGP protocol, these four message types had been defined. The BGP protocol is designed to be flexible, which means that it can support the addition of new message types through future extensions. However, since the time of writing this thesis, only one new BGP message type has been standardized. This new message type is called the **Bgp Route-Refresh** message [Che00]. It allows the router to ask its neighbor to re-announce the routes stored in the Adj-RIB-out, which are maintained specifically for the related router.

BGP Route Attributes

In addition to the AS-Path, each BGP route is advertised with attributes that allow routing policies to be applied and enforced. The BGP standard defines several attributes which are divided into several categories:

- The **Well-known mandatory attributes**. These attributes are essential and must be processed and included by every BGP router when it advertises routes. The AS-Path, which indicates the ordered list of ASes that a route takes, is one such attribute. Other required attributes are Origin, which indicates whether the route was learned from an IGP, EGP, or other source, and Nexthop, which indicates the next router to forward IP packets to the advertised IP prefix.
- The **Well-known discretionary attributes**. This type of attribute is understood by all BGP implementations, but they are not required to be present in every BGP route.
- **Optional attributes**. BGP is a flexible protocol that supports the addition of new extensions, including new attributes. However, not all

routers support the same set of features. As a matter of fact, some attributes defined in these extensions cannot be recognized by the implementation. Hence, to allow the exchange of these attributes, BGP allows them to be defined as optional. There are two types of optional attributes. First, there are **transitive optional attributes**. If a router receives an attribute of this type but does not recognize it, it will still accept it and pass it on to other BGP routers. Conversely, there are **non-transitive optional attributes**. If the implementation does not recognize it, the attribute will not be forwarded to other routers and will be removed from the original route.

The BGP Decision Process

This is probably the most important mechanism in BGP since it is in charge of selecting the best routes. The router applies a list of methodical rules to differentiate the routes to the same IP prefix. If a rule fails to make a decision, the router proceeds to the next rule and so on until a single route is chosen among all the available ones. We detail each of these rules in the order they are applied by BGP. Over time, other rules have been added to the original BGP decision process thanks to the introduction of new extensions. To not complicate the process, the explanation only covers the original set of rules defined during its standardization.

1. Ignore routes with unreachable Nexthop.

   This first step checks the reachability of the Nexthop specified in the BGP update messages. BGP checks whether the Nexthop address is found in the routing table and can be reached. This means that there must be at least one valid route in the routing table that can reach the specified Nexthop.

   The Nexthop can be reached in a number of ways, including a directly connected route, a static route or a route learned from the IGP. If the address of the Nexthop is not accessible on the basis of the routing information available, BGP does not take it into account in the decision process.

   This behavior is important to avoid traffic blackholing, where packets are lost or discarded without reaching their intended destination. By ensuring that only routes whose Nexthop is reachable are taken into account, BGP guarantees that traffic is routed along valid and functional paths.

2. Prefer routes with the highest Local-Pref.

The Local-Pref attribute is a numerical value assigned by the network operator to indicate the preference for a specific route over others. This attribute is not transmitted over eBGP sessions. Instead, it is a local attribute used within an autonomous system (AS) and between iBGP sessions. It plays a significant role in determining the egress link, which determines the AS to which IP packets will be forwarded.

For instance, when an operator receives the same route from both an American AS and a European AS, it may prefer one over the other. The Local-Pref attribute allows operators to exert control over which route is favored for outgoing traffic. By assigning a higher Local-Pref value to a particular route, the operator indicates a stronger preference for that route and influences the traffic flow accordingly.

3. Select the route with the shortest AS-Path.

   The AS-Path refers to the sequence of ASes that a BGP route traverses from its origin to the destination. BGP prefers routes that have a shorter AS-Path, as it signifies fewer AS hops or intermediate networks to reach the destination. This rule aligns with the principle of efficiency, as shorter AS-Paths generally indicate a more direct and optimal routing path.

   By prioritizing routes with shorter AS-Paths, BGP aims to minimize the number of ASes traversed and reduce potential points of failure or congestion along the route.

4. Prefer routes with the lowest Origin

   The Origin attribute is represented by a numeric value and provides information about the origin of a BGP route. Specifically, it indicates whether the route originated from an IGP (Interior Gateway Protocol), EGP (Exterior Gateway Protocol), or if the origin is unknown.

   - Routes with an Origin value of 0 indicate that they originated from the IGP.
   - Routes with an Origin value of 1 indicate that they were learned from the EGP.
   - Routes with an Origin value of 2 indicate that no information about their origin is available (Unknown).

   In accordance with this rule, BGP favors routes that originate from the IGP (Origin value of 0) over routes learned from the EGP (Origin value of 1). Additionally, routes with an unknown origin (Origin value of 2) are considered less favorable compared to those with a known origin.

This rule prioritizes the selection of routes that have a clear and trusted origin within the network.

5. Prefer routes with the lowest Multi-Exit-Discriminator (Med)

   The Med is a numeric value used to differentiate routes originating from the same AS. In situations where multiple links exist between ASes, the same routes may be advertised on those links. The Med value is used to indicate the ingress preference to the neighboring AS.

   Preferring the lowest Med value may be counterintuitive. However, the Med attribute was originally designed to reflect the IGP distance to the destination. A lower Med value means that the IGP distance to the destination is shorter.

   However, because of the convergence problems it can cause, such as oscillations [GW02a], some network operators choose to disable the use of the Med attribute. Its behavior can be unpredictable and lead to routing instability. For this reason, some BGP implementations disable the Med attribute by default.

   The use of the Med attribute is optional and depends on the network operator's configuration and preferences.

6. Prefer routes learned from an eBGP session to an iBGP session.

   Operators generally favor fast forwarding of packets in transit to their destination. This approach, often referred to as "hot potato," aims to minimize transit time and reduce the number of intermediate routers involved in forwarding packets.

   By favoring routes learned over eBGP sessions, which are established with neighboring AS, over routes learned over iBGP sessions within the same AS, operators can quickly forward packets in transit to external networks. This minimizes the equipment required for routing and promotes efficient and direct routing paths.

   The goal of this rule is to speed up the delivery of traffic by quickly forwarding it to external networks, allowing faster transit through the network and reducing unnecessary internal hops.

7. Prefer routes with the lowest IGP metric to the BGP Nexthop

   The motivation is the same as for the previous rule. Network operators prefer to forward packets using routes within their own AS which have the lowest IGP metric value when it concerns packets to external destinations. This preference for low IGP metric routes ensures that packets are forwarded as fast as possible to other ASes. This is essentially

a strategy to optimize internal routing within an AS when external traffic is involved.

8. Prefer routes learned by the lowest Router-Id

   The Router-Id is a unique identifier associated with each BGP router in the network. It is an arbitrary value selected by the network operator and typically corresponds to one of the IPv4 addresses assigned to the router. The Router-Id is encoded as a 32-bit value.

   While this rule does not directly apply a routing policy, it serves as a fallback mechanism to break any remaining ties when other rules cannot differentiate between routes. By selecting the route with the lowest Router-Id, BGP ensures a definitive choice among competing routes.

   Choosing the lowest Router-Id is only considered as a last resort, typically when no other decision criteria can determine the best route.

9. Prefer routes received from the peer with the smallest IP address.

   The penultimate rule may not decide the route. In the case of iBGP sessions, where routes are exchanged within the same AS, this rule helps determine the preferred route when other criteria have not been sufficient. BGP relies on the IP address of the router that sent the route as a final deciding factor.

   By comparing the IP addresses, BGP can identify the route associated with the router having the smallest IP address among its peers.

   This rule is effective in distinguishing between routes because each BGP neighbor is assigned a unique IP address. It serves as a last-resort measure to select the "best" route when all other decision criteria have been exhausted.

1.6 Routing Security

Given the current size of the Internet, routing incidents, whether unintentional or malicious, are common and can disrupt a significant portion of the network. The importance of routing security has increased dramatically as the Internet has evolved from a small network to a tool that complements our daily lives. To address this issue, several collectives and initiatives have emerged, such as the Mutually Agreed Routing Security Standards (MANRS)[1]. MANRS brings together Internet service providers, Internet exchange points (IXPs),

[1] https://www.manrs.org/

cloud service providers, and other stakeholders to establish best practices for securing routing on the Internet.

A notable initiative of MANRS is the development of an interactive website that allows users to view routing incidents occurring on the Internet each month[2]. This site provides valuable information on the frequency and impact of routing incidents. In May 2023, more than 1,500 routing incidents have been reported, highlighting the importance of the problem.

This section outlines the various methods an operator can use to prevent routing issues. There are two key aspects of routing security: transport security and the integrity and authenticity of the content of routing messages. Taking these aspects into account helps prevent routing problems and improves the overall security of the Internet's routing infrastructure.

1.6.1 Securing the transport of routing messages

In shared environments such as Internet Exchange Points (IXPs), routers from different Autonomous Systems (AS) operate in an uncontrolled and untrusted environment. This introduces risks of router misconfiguration or malicious activity that can compromise BGP sessions between routers. Similarly, IGP protocols within internal networks may also be vulnerable to malicious attacks. To address these concerns, this section focuses on the main methods used to secure the transport of routing messages.

TCP Reset. One possible attack is the TCP Reset attack, described in RFC 3360 [Flo02]. In this attack, an attacker can maliciously reset the BGP session between two other routers and establish a new unauthorized session with one of the affected routers.

Routing Message Authentication. To secure the transport of routing messages, the protocols themselves do not provide encryption for data sent between routers. Instead, the data is transmitted in clear text but authenticated using a pre-shared key and a cryptographic function called HMAC (Hash-based Message Authentication Code) [KBC97]. OSPF uses HMAC-SHA [Fan+09], while IS-IS uses HMAC-MD5, [Man+09]. In this process, the contents of the routing message and the pre-shared key are hashed using the specified cryptographic function. The resulting hash is stored in a specific field of the routing message. When it receives the message, the destination router performs the same calculation and compares the calculated hash with the one included in the routing message. If they match, the message is considered authenticated and accepted.

[2] `https://observatory.manrs.org/#/overview`

In the case of BGP, the approach is different because TCP provides inherent support for authenticating TCP segments. BGP routers can configure the TCP stack to authenticate the transport using mechanisms such as TCP-MD5 [Hef98], or TCP-AO (TCP Authentication Option) [TBM10]. With these mechanisms, it is not necessary to modify the content of the BGP messages. BGP routers simply configure the TCP stack to authenticate TCP segments.

The HMAC authentication technique for routing messages does have certain drawbacks that need to be considered. First, one major concern is the obsolescence of the MD5 algorithm, which has been deemed insecure since 2004 [WY05]. Using MD5 for authentication can no longer guarantee strong cryptographic security. Therefore, relying on MD5 for HMAC authentication in routing protocols poses a significant risk.

Second, in the case of IGP protocols such as OSPF and IS-IS, all routers in the network may share the same pre-shared key, which increases the risk of key leakage or compromise. If the shared key falls into the wrong hands, it can lead to unauthorized access and manipulation of routing information, which can cause serious disruptions. Of course, operators can use a different key for each pair of routers in the network. This approach ensures that even if one key is compromised, it will not affect the security of other router pairs. However, managing and maintaining a separate key for each router pair in a very large network can become unscalable and administratively burdensome.

The management overhead includes securely distributing and updating the keys across all routers, ensuring that keys are properly stored and protected, and handling key rotation or revocation when necessary. With a large number of routers, these tasks can become complex and time-consuming for network operators.

To address scalability issues, network operators can opt for hierarchical key approaches. In these approaches, routers are organized into groups or domains, and a different key is used for each group rather than for each pair of routers. Network operators have various solutions at their disposal to ease the deployement of a such strategy, such as Vault by HashiCorp [Has15] or Oracle Key Vault [Ora15]. This reduces the number of keys to manage while providing a reasonable level of security. Each router in a group shares the same group key, simplifying key distribution and management.

Another approach to scalability is to use key management protocols, such as the Internet Key Exchange (IKE) protocol [FK11], which automates the key establishment and distribution processes. These protocols provide secure and efficient key exchange mechanisms, reducing the administrative burden associated with managing individual keys for each router pair.

Last, for a considerable period of time, TCP-MD5 [Hef98] was the only available option for authenticating BGP sessions. However, TCP-AO was introduced as an alternative in 2010 [TBM10]. Unfortunately, it took a significant

amount of time for vendors to implement TCP-AO in their operating systems. Only in 2020 did vendors like Nokia, Juniper, and Cisco begin supporting TCP-AO in their systems [Ael20]. This delayed implementation meant that network operators had limited options for securing BGP sessions during that time.

Although there are many cryptographic methods for authenticating routing messages, the evolution of routers is slow and does not necessarily include all features. The support for new features designed to improve BGP security is dependent on their implementation by router vendors, as well as by the operators who need to use them in their networks.

Secure Tunnels. By encapsulating OSPF routing messages in these secure tunnels, operators can ensure that the information exchanged between routers remains confidential, protected from eavesdropping and unauthorized routing message injection. IPSec [FK11] and Wireguard [Don17], two widely adopted protocols for secure tunnels in network environments, provide strong encryption, authentication and key management features. The use of encrypted tunnels is particularly prevalent in multi-site ASes that are not directly connected to each other. In such scenarios, routing messages must traverse the Internet, which presents potential security risks. By leveraging encrypted tunnels, network operators can mitigate these risks and ensure confidentiality and integrity of routing messages as they traverse untrusted networks.

Data-plane protections. In addition, there are security features available in the data plane that use IP or Ethernet filters. These filters allow operators to restrict routing sessions to specific authorized addresses. By configuring firewalls, operators can ensure that only a selected portion of their infrastructure is allowed to establish connections with their routers. This practice enhances security by limiting access to trusted entities and minimizing the possibility of unauthorized routing sessions.

However, this approach is not resilient. If a network failure disables the interfaces authorized by the filters, control messages may not be received, as they may be blocked on other interfaces due to the filters. It is therefore essential to use filters with caution, considering all possible failure cases.

Setting the Time To Live field. Routing sessions typically take place over point-to-point (P2P) links, where two directly connected routers establish a direct communication path without passing through other routers. The Generalized TTL Security Mechanism (GTSM) [Pig+07] exploits this feature to improve routing security. According to GTSM, routing messages must have a Time To Live (TTL) value of 255. The TTL is an 8-bit field in the IP

header [Inf81] that decreases by 1 as the packet is transmitted by each router. If the TTL reaches 0, the router discards the packet. By setting the TTL to the maximum value of 255, routers can determine whether a packet came directly from the P2P link or was received from another source. If a router receives an IP packet with a TTL less than 255, it indicates that the packet was not sent directly from the neighboring router and should be discarded for security reasons.

1.6.2 Securing the authenticity of routing messages

When a router receives a routing message announcing a new IP prefix, it is not certain that the route is valid and has not been changed in transit. In the context of an IGP, where routing is limited to a single AS, the network operator has control and knowledge of the network, which reduces the risk of route falsification. However, in the case of an EGP, such as BGP, no assumption can be made about the legitimacy of an AS. The remainder of this section discusses the major threats that can arise when exchanging BGP routes and the security measures that can be applied.

Route Origin Hijacking Every year, ASes deliberately or accidentally advertise IP prefixes that do not legitimately belong to them. A notable incident occurred in 2008 when Pakistan Telecom blocked access to YouTube for the entire Internet, falsely claiming, for censorship purposes, that its AS was the legitimate owner of the prefix containing YouTube's DNS servers.

The lack of a mechanism to certify the legitimacy of BGP route advertisements is a significant challenge to routing robustness. A single compromised AS can potentially disrupt the entire routing system. To address this problem, the BGP protocol relies on the Resource Public Key Infrastructure (RPKI) [LK12] for resource certification. RPKI allows an AS to cryptographically authenticate its ownership of resources, thus providing assurance to BGP routers.

RPKI is based on the X.509 certificate system [CCI89; Boe+08], which includes additional fields to represent network resources. X.509 certificates use asymmetric cryptography [DH76] and contain a public key and information about the certificate holder, called the "subject name." These certificates can be customized with different fields encoded in ASN.1 format [CCI84; CCI88]. Certificate generators such as OpenSSL [The03] allow the inclusion of specific information, such as IP addresses or alternative domain names, as described in §4.2.1.6 of RFC5280 [Boe+08].

Certificates are used to verify the authenticity and integrity of information exchanged over networks. The information contained in a certificate is signed using the private key of a trusted Certificate Authority (CA), and the signature

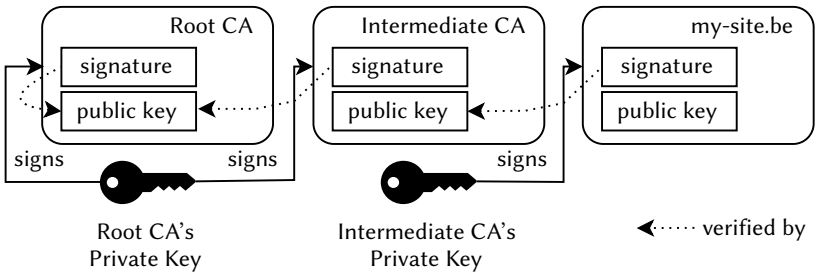


Figure 1.5: Example of certificate chaining.

is stored in the certificate itself. The best-known certification authorities in the web world are GlobalSign, Digicert and Verisign, which are called "Root CAs".

In practice, not all certificates are issued directly by these root CAs. Instead, they delegate the certification process to sub-certifying entities, such as the company or organization that owns the resource to be signed, or a subcontractor. This forms a certification chain, as shown in Figure 1.5. For example, the certificate of a website such as `my-site.be` can be signed by one or more intermediate certification authorities, which are themselves signed by the root certification authority.

When a web browser connects to a website, it receives the website's certificate from the server and begins validating the certificate chain. The browser verifies the signatures of all certificates using the public key associated with the private key that signed each certificate. The browser also verifies other critical information contained in the certificate, e.g., the validity date of the certificate, the domain name, etc. This validation process continues until it reaches the root certification authority. If all signatures are valid and the certificate information is verified, the certificate is considered valid. This ensures that the web client is communicating with the intended website (`my-site.be`) and not a malicious site.

Root certification authorities, which are trusted authorities, have self-signed certificates. This means that the private key associated with the public key of the certificate is used to sign the certificate itself. To allow browsers to verify certificates, root certificate authorities are embedded in the web client or operating system. This prevents the installation of malicious root certificates. However, users must trust their web browser or operating system, as trust in root certificate authorities is inherited from these software platforms.

Unlike the web world where servers are authenticated, BGP focuses on authenticating IP resources and ASes. The certification chain and actors involved in BGP authentication are therefore different. In the context of BGP, the Resource Public Key Infrastructure (RPKI) is currently used to certify the

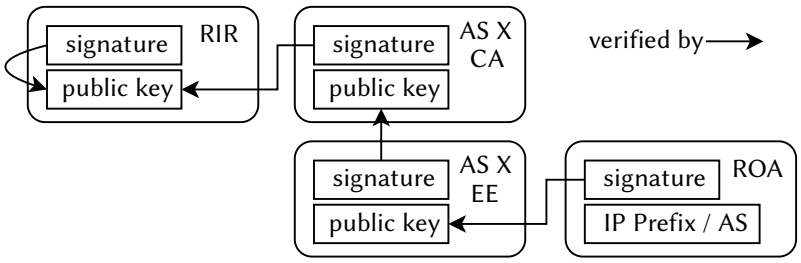


Figure 1.6: Chain of trust for ROAs.

route origin, ensuring that the advertised IP prefix is originated from the legitimate AS that holds it.

The certification chain in BGP authentication involves the use of Route Origin Authorizations (ROAs) [LKK12] that are digitally signed and authenticated. ROAs are files formatted in the Cryptographic Message Syntax (CMS) [Hou09] using ASN.1. Figure 1.6, similar to Figure 1.5, illustrates the certification chain used for the digital signing and authentication of resources.

In the ROA certification chain, the ROA containing an Internet resource is signed by the End Entity (EE) X.509 certificate from the AS that owns the resource. The EE certificate is specifically used for signing objects like ROAs and is not used for signing other certificates. The rest of the certification chain remains the same, where intermediate CAs owned by the AS sign the EE certificate, and the root CAs sign the intermediate CA certificates.

The Regional Internet Registries (RIRs), like RIPE NCC, APNIC, and AFRINIC, maintain the root CAs of the RPKI system. Their management and maintenance are the key element to the RPKI system.

Each RIR maintains its own database for the IP resources it manages. For example, if a Belgian university wants to participate in the RPKI system, its route origin authorizations (ROAs) will be likely signed by the RIPE NCC, the RIR responsible for managing Internet resources in Europe.

The RIRs are considered trusted anchors and provide repositories where signed objects are accessible. These repositories are available using protocols such as remote synchronization (rsync) [TM96] or the more recent RPKI Repository Delta Protocol (RRDP) [Bru+17]. To reconstruct the complete RPKI database, which contains all RPKI objects for the entire Internet, one would have to contact the repositories of all RIRs. However, using a BGP router to perform this task is not ideal. Signature verification involves computationally intensive cryptographic algorithms for which routers are generally not optimized.

To solve this problem, RPKI caches are distributed throughout the Internet. These caches are dedicated servers that store copies of signed objects. Routers

can query these cache servers to verify BGP advertisements. RPKI cache servers have three main roles. First, they obtain copies of all signed objects from the repositories. Second, they perform the objects' validation by verifying their signatures up to the root CA. Third, when the cryptographic objects have been validated, the RPKI cache builds a trust base that contains for each IP prefix, the originating AS.

Finally, to transmit the trust base of the RPKI cache to the BGP router, the RPKI system uses the RPKI to Router (RTR) protocol [BA17]. RTR is a lightweight protocol used for communication between a BGP router and an RTR enabled server (i.e., the RPKI cache). Once a connection is established, the RTR server sends the router the trust base, which contains validated <AS, IP prefix> pairs. The RTR protocol supports various communication protocols such as SSH or TCP.

When a BGP router receives a BGP advertisement, it checks whether the <AS, IP prefix> pair in the update matches an entry in the trust database. BGP announcements can be classified into three states. The first state is **Valid**, which means that the advertisement has been cryptographically verified and matches an entry in the trust database. The second state is **Invalid**, which means that the prefix was found in the database, but no <AS, IP prefix> entry matches the BGP advertisement. The third state is **Unknown**, which means that no <AS, IP prefix> pair was found in the trusted database.

The RPKI ROA validation specification does not specify how the router should react to Invalid or Unknown prefixes. The specific actions taken by the router in these cases are determined by the network operator's policies. As a general rule, Invalid prefixes are rejected because they do not have a valid match in the trust database. Unknown prefixes, on the other hand, are often accepted but assigned a lower Local-Pref, in the hope that another valid <As, Ip prefix> advertisement will match the trusted database and be preferred.

AS-Path Manipulation While route origin validation provides a way to verify the legitimacy of the origin of IP prefixes in BGP advertisements, it does not completely solve all the problems associated with BGP. One such problem is AS-Path manipulation, where intermediate ASes can be modified to divert or disrupt traffic. To solve this problem, BGPSec [LS17] introduces cryptographic signing of the whole BGP path.

In the traditional BGP protocol, when a router announces a route to its neighbor, it includes its own AS and the neighbor's AS in the AS-Path attribute. In the BGPSec protocol, this AS-Path attribute is replaced by the BGPSec-Path attribute, which contains a cryptographic signature. The AS originating a BGP announcement signs the current BGPSec-Path with its private key. The public key corresponding to the signing AS is published in the RPKI certificates, which are part of the RPKI system.

When a router receives a BGP advertisement with the BGPSec-Path attribute, it can validate the signatures in the attribute. By checking the entire path, including all involved ASes, the router can determine the legitimacy of the advertisement. If the signatures are valid for the entire path, the advertisement is considered legitimate and can be accepted.

For BGPSec to provide robust security, a global participation is fundamental. This means that all BGP routers on the Internet must support the BGPSec extension [LGS13]. Router vendors must also ensure that their hardware is capable of supporting BGPSec and operators must update their routers accordingly. In addition, operators must publish their certificates in the RPKI databases. These requirements imply a significant change in the way BGP announcements are currently handled on the Internet.

Route Leaks BGP peering links between ASes are primarily established through political and commercial agreements. The most common type of agreement is based on the "provider-client" principle. In this case, the **provider** AS is responsible for announcing all the routes it knows to its **client** AS. On the other hand, the client AS only announces its own prefixes and those learned by its own clients to the provider.

In some cases, two ASes have a mutual agreement and share links, forming a **shared link** relationship. In this case, both ASes advertise their own routes and those learned by their respective clients.

However, sometimes an AS mistakenly announces routes that are beyond its intended scope, which is called a route leak [Sri+16]. Route leaks are typically caused by configuration errors in the import and export filters of routers.

Currently, there is no concrete standard for protecting against route leaks. However, there are two main approaches to solve this problem.

The first approach involves the use of monitoring tools that track changes in BGP (Border Gateway Protocol) announcements in real time. If a route leak is detected, the monitoring tool immediately informs the network operator. The operator can then take manual action to manage the incident and quickly remedy the problem.

The second approach, currently being standardized by the IETF, uses RPKI to detect route leaks. In addition to ROA records, a new cryptographic object called ASPA (Autonomous System Provider Authorization) is integrated into the RPKI system [Azi+23b; SA23; Azi+23a]. The ASPA object allows ASes to specify which other ASes are authorized to advertise their routes. When a route is advertised, the BGP router compares the AS-Path received and checks the validity of each AS pair against the ASPAs.

1.7 Beyond traditional distributed routing

In the previous sections, we discussed existing distributed routing protocols, such as BGP, OSPF, ISIS, and EIGRP, which play a key role in enabling the Internet to learn and distribute network routes. These protocols have been widely adopted and are responsible for network connectivity and reachability.

However, one of the limitations of these traditional routing protocols is their rigid and inflexible nature. They have evolved over time and have become somewhat "ossified", meaning that they have reached a stable state and are resistant to significant change or modification. The ossified nature of these protocols can be attributed to their wide deployment, interoperability considerations, and the need for backward compatibility. As a result, introducing substantial changes or new features into these protocols can be difficult and time-consuming. Network operators may face limitations when attempting to implement advanced routing policies, fine-grained control, or dynamic network behavior that goes beyond the capabilities provided by these protocols.

To address these limitations and enable more flexible network management, researchers and industry professionals have explored alternative approaches such as **Software-Defined Networking** (SDN) [McK+08]. SDN decouples the control-plane from the data-plane, allowing operators to centrally control and program their networks through a software-based controller. This provides greater flexibility, programmability and the ability to define network behavior based on specific requirements.

In the SDN architecture, the control plane is centralized on a device called a controller. This controller maintains a global view of the network and communicates with the network devices, which act as specialized switches with only data plane functionality, meaning that they are only able to forward packets. The controller interacts with the switches through an API such as OpenFlow [McK+08], which allows it to define forwarding rules and manage network behavior.

Centralizing the control plane simplifies network management compared to traditional distributed routing protocols. The controller can perform advanced routing operations and implement custom policies that go beyond the limitations of distributed protocols. For example, SDNs enable the implementation of per-flow security policies and dynamic load balancing.

Despite the benefits of SDN, its widespread deployment remains limited in ISP networks. The introduction of SDNs into networks presents challenges in terms of scalability and resiliency. Because the controller is responsible for the entire network, it can be overloaded when managing large-scale networks with thousands of switches. In addition, the controller becomes a single point of failure, as the operation of the network depends entirely on its availability. Finally, in some network configurations, the SDN controller may have a

different view of the network than the routers. For example, in Hybrid SDN approaches [VVB14], one part of the network operates according to SDN principles, while another part relies on traditional distributed networking methods. When a failure occurs on either side of the network, effective coordination between these two parts may be limited, resulting in prolonged downtime.

Addressing scalability and resiliency issues is an ongoing area of research and development in SDN. Various techniques, such as distributed controllers [BSM18], fault-tolerant designs [Bot+14] and robust deployment of Hybrid SDN [ARS18] are being explored to mitigate these issues and make SDN a more practical and robust solution for network management.

Part II

Bringing innovation back in routing with truly extensible protocols implementations

The Need of Extensibility in Routing Protocols

2

During the last decades, the requirements imposed on enterprise and Internet Service Providers (ISP) networks have changed drastically. The first enterprise networks simply provided a "best effort" service and were not attached to a public network. Today's enterprise networks need to support Quality of Service [EF10] and include security feature to protect them from attacks originating from the Internet. ISP networks also face similar problems, but at a much larger scale [Hus98; Dav04]. Internet traffic continues to grow quickly and ISP networks need to scale to sustain the load. Indeed, ISPs are continuously challenged by their users and customers to provide value-added services that go beyond best-effort connectivity. Among others, these new services include traffic engineering techniques to prioritize some flows over others and improve network load, fast reroute mechanisms to swiftly retrieve connectivity upon failures, or anycast routing. In addition, ISPs are trying to improve their internal operations in order to provide an ever better service to their customers. This can be done by implementing a monitoring system, re-architecting or tuning the internal network.

To cope with these changing requirements, network operators are forced to innovate. **Innovation** is defined by the Merriam-Webster dictionary as the *introduction of something new.* As an extension, we can define *network innovation* as the introduction of new features inside an enterprise or ISP network. The introduction of a new feature is often done in three phases: *design*, *implement* and *deploy*. During the *design* phase, the network operators propose new abstract solutions and evaluate them. One of the proposed solutions is then implemented before being deployed it in the network after successful tests in labs.

Innovation almost always requires the extension of routing protocols. And among all protocols, the Border Gateway Protocol (BGP) is probably the most used one given its flexibility: for many network operators, BGP has become a true "Swiss-army knife". Originally designed to distribute interdomain routes, BGP has been extended several times to support different types of services [KKC12; RR06].

While extending BGP is possible, it is certainly not easy, for two main reasons. First, ISP networks often include routers from different vendors [Van+13;

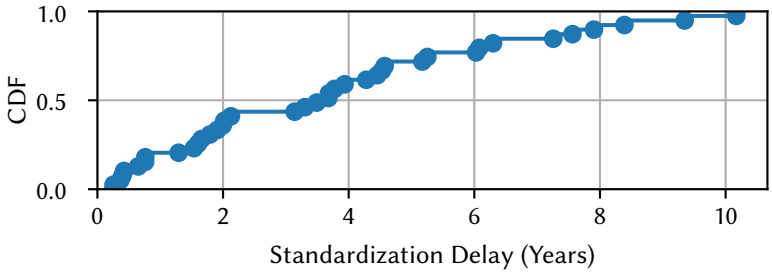


Figure 2.1: Delay between the publication of the first IETF draft and the published version of the last 40 BGP RFCs in 2020.

Dav04]. This diversity is inherent and required for technical, safety, economic reasons, and the type of use. The characteristics of a software-based virtual router used in a datacenter are different from those of an access router that connects a remote branch office or a backbone router that supports hundreds of terabits/sec of traffic. Unfortunately, this diversity means that operators can only use the *intersection* of the features set across all their routers, hindering flexibility. Also, some vendors only supply certain types of routers (e.g., access or backbone ones). To prevent problems from software bugs, some operators connect important customers to two routers from different vendors to ensure that they would not fail at the same time. From an economical viewpoint, sourcing the network vendors from different manufacturers increases the competition among them.

Second, it can take years for even a subset of the vendors to implement new features as these need to be first standardized by the Internet Engineering Task Force (IETF). Many view this as a form of ossification of the routing protocols. As an illustration, Figure 2.1 depicts the delay between the moment the IETF working group responsible for the BGP routing protocol (IDR) started to work on a new feature and its actual RFC publication. This delay includes the time required to document two independent and interoperable implementations as required by the IETF Routing Area. We see that the median delay before RFC publication of BGP extensions is *3.5 years*, and that some features required *up to ten years* before being standardized[1]. This long delay has also been confirmed by another study [McQ+21]. This is only the tip of the iceberg, though: only a small subset of the BGP extensions proposed by network operators have been discussed and later adopted by the IETF.

Another difficulty for our network operator is that each vendor has a different configuration language. BGP can be tuned using various access lists, filters, but they are all specific to each vendor. Some network operators have

[1]Note that this delay ignores the time elapsed between the initial idea and its first adoption by the working group, making the actual delay even longer.

developed tools to automatically translate the most frequent BGP configurations in different vendor languages [Got+03], but these tools do not support all BGP knobs. The vendor-neutral management interfaces developed within the IETF starting from SNMP [HH06] to the YANG models used by `netconf` [Jet+21; Qu+21] only support a subset of the router features.

Frustrated by these delays and the difficulty to innovate in networks, researchers have argued for Software-Defined Networks (SDN) [McK+08] for more than a decade. Instead of relying on a myriad of distributed protocols and features, SDN assumes that switches and routers expose their forwarding tables through a standardized API. This API is then used by logically centralized controllers to "program" routers and switches. By using centralized controllers that program flow-tables on the network switches and routers, researchers and network operators can implement a wide range of services that are difficult to support with traditional routing protocols (see the references cited by Kreutz et al. [Kre+14] for a long list of examples). Some companies rely on SDN for parts of their network [Jai+13], but SDN has not replaced traditional routing protocols like BGP and OSPF/IS-IS.

While SDN has enabled countless new research works [FRZ14; Kre+14], it has not been widely adopted by ISPs. One of the main hurdles is that deploying SDN requires a major network overhaul, both at the control-plane level, to deploy scalable and robust logically-centralized controllers, *and* at the data-plane level, to deploy compatible network devices. Thus far, only large cloud providers managed to perform this overhaul [Jai+13; Hon+13].

Of course, instead of relying on commercial routers, network operators could decide to adopt open-source implementations of routing protocols [CZN20; Fou17; HHK03; The] running on servers or custom hardware [Mica]. A network operator could for instance fork a BGP implementation to add a desired feature. Maintaining this fork requires a lot of software development effort, though. Such an approach is feasible for large cloud providers [Sin+21] but not for ISPs. Another approach is to use a modular routing implementation to take full control of the protocol. The network operator is responsible for the entire routing implementation. Unfortunately, it is too difficult to maintain and evolve because the network operator must have a complete understanding of the routing protocol and must have software programming skills, which they often do not have. To provide flexibility in the administration and automation of their routers, some router vendors have added a Python interpreter to their operating systems [Jun21]. However, the interpreter only handles the administration part of the router and does not provide an interface to add or modify protocol features. Finally, in the late 1990s, active networks were proposed as a solution to bring innovation back within networks [TW96]. Several architectures were proposed to enable routers to execute bytecode encoded inside packet headers. Prototypes were

developed [Ten+97], but active networks were not adopted by industry [Cal06]. Indeed, the use of active networks with centralized approaches or descriptive configuration languages [Cae+05; GS05] is not possible in today's Internet, as autonomous systems still use decentralized protocols to establish peering links. Therefore introducing arbitrary operations directly into the network poses a serious security problem that needs to be solved.

The Difficulty of Deploying New Features in a Network

In practice, deploying a new service inside a large ISP network can be difficult. As previously explained, such a network is rarely composed of homogeneous routers produced by the same manufacturer. Still, these different routers support the same packet format (IP) and implement the same standardized routing protocols (OSPF [Moy91], BGP [RL94]).

To illustrate the difficulty of deploying extensions and new services to routing protocols, let us look back at the evolution of several BGP and OSPF extensions whose deployment has been documented.

The BGP communities [TSR06] play an important role in scaling BGP routing policies by enabling network operators to tag routes and then apply the same policies to the routes that carry a given tag. Various use cases of this attribute have been documented [DB08; Gio+17; Str+18]. This BGP attribute was designed when BGP used AS numbers encoded as a 16-bits integer and the high-order bits of the BGP communities contain an AS number. As BGP evolved to support 32-bits AS numbers [VC12], it became necessary to support wider BGP communities. Since 2009, the new ISPs receiving 32 bits AS numbers by default were unable to define their own BGP communities according to the existing standard [TSR06]. Several encoding for BGP communities that support 32-bit AS numbers were proposed since 2002 [And+10; Ras+18]. Unfortunately, discussions did not converge within the IETF and it took more than fifteen years to finally agree on the BGP Large Communities specification [Hei+17]. The redaction of this document took less a than year, probably a record for the IETF and implementations were released in the following two years[2]. The first ISP that received a 32-bits AS number in 2009 had thus to wait more than a decade to be able to use its assigned BGP communities.

Another example is the BGP extensions to support the traffic engineering performance metrics [Gin+19]. The development of these extensions started in 2013 [Wu+13]. Six years later, it was supported by only two vendors[3].

[2] See `http://largebgpcommunities.net/implementations/` for a detailed description of these implementations.

[3] See `https://trac.ietf.org/trac/idr/wiki/draft-ietf-idr-te-pm-bgp%20implementations`.

A third example is the so-called add-path BGP extension [Wal+16] that enables a router to send several paths towards the same prefix over a single BGP session. The first discussions for this extension started in 2002 [Wal+02] and the IETF approved it in 2016. The first implementations were reported in 2011 and then mainly in 2014[4].

A fourth example is the OSPFv3 LSA extensions [Lin+18] that extend the LSA format by encoding the existing OSPFv3 LSA information in Type-Length-Value (TLV). Thanks to these TLVs, it becomes easier to use OSPFv3 to flood other types of information than those covered by the standardised LSAs. The first discussions on this extension started in 2013 and only two implementations have been confirmed[5].

These examples show that while standardised routing protocols have clear benefits in terms of interoperability, it often takes half a decade or more before network operators can deploy new network services that require protocol extensions. Large companies like Facebook have reacted by implementing their own proprietary routing protocol [Has+], but there are no indications of its adoption outside Facebook.

In the two following chapters, we propose a compromise that combines the flexibility of SDN, enabling network operators to implement their own code, with the benefits of distributed routing protocols, without the need for hardware support by relying on specialized switches as required with SDNs. We focus on BGP, but the solution that we propose is applicable to other routing protocols with some implementation effort. Similarly to what OpenFlow [McK+08] achieved, we believe that programmable distributed routing protocols have the potential to open up *many* promising avenues for research, while being fundamentally more practical and deployable.

[4] See `https://trac.ietf.org/trac/idr/wiki/draft-ietf-idr-add-paths%20implementations`.

[5] See `https://trac.ietf.org/trac/ospf/wiki/draft-ietf-ospf-ospfv3-extend%20implementations`

Extending routing protocol implementations with plugins

3

This chapter is largely based on the paper T. Wirtgen, C. Dénos, Q. De Coninck, M. Jadin, and O. Bonaventure. "The Case for Pluginized Routing Protocols". In: *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE. 2019, pp. 1–12. DOI: 10.1109/ICNP.2019.8888065.

Network operators typically depend on commercial implementations of closed-source routing protocols, mainly because they can obtain support and updates from the vendor in the case of issues. These implementations are like black boxes in that their inner workings are hidden and only configurable and accessible via router interfaces such as the command line interface (CLI), SNMP or NetConf.

Consequently, operators are limited to the predefined functions and configurations provided by the router vendor. They cannot modify or customize the underlying protocol code, which prevents them from adapting routing protocols to their specific needs or exploring new approaches to improve their network. In addition, the lack of a standardized mechanism for modifying or extending protocols worsen the situation. Network operators have to go through a complex and cumbersome process to propose changes through organizations such as the IETF (Internet Engineering Task Force). This challenging process further complicates the already time-consuming task of introducing improvements or innovation in routing protocols.

As a result, the potential for innovation within networks is severely limited, as the lack of flexibility throttles innovation and prevents operators from taking advantage of emerging technologies or adapting to changing network needs. They are faced with the inherent limitations of existing protocol implementations, unable to explore alternative routing strategies or integrate state-of-the-art solutions that could optimize network performance, enhance security, or enable more efficient traffic management.

To solve this problem, the first part of this thesis's section aims to explore the possibility of dynamically extending a routing protocol implementation

with executable code written by the operator. We use a virtual machine based on eBPF, capable of executing code written in the C programming language.

The rest of this chapter is organized as follows. We first start to explain the eBPF environment in Section 3.1. We then propose **three main contributions**:

- First, we propose in Section 3.2 to organize the implementations of routing protocols so that a network operator can extend the implementations used on her routers to support new protocol features.
- Second, we demonstrate that such an architecture can be implemented in the OSPF and BGP daemons of FRRouting in Section 3.2.1.
- Third, we demonstrate in Section 3.3, several use cases showing the benefits of our proposed implementation architecture.

We continue by exploring the related work of network programability in Section 3.4. We finally, conclude this chapter in Section 3.5.

3.1 The eBPF environment

This section provides a brief explanation of eBPF and how it enhances the flexibility of routing protocol implementations. eBPF, or extended Berkeley Packet Filter, is a versatile technology that allows custom code to run securely within the Linux kernel through the eBPF virtual machine (VM), enabling flexibility and safe execution.

eBPF was originally developed specifically for the Linux kernel. It is built by extending BPF (Berkeley Packet Filter), a lightweight virtual machine used for simple rule-based network packet filtering [MJ93]. The idea of extending BPF emerged in 2014 when Linux developers were looking to reduce the amount of code interpreted in Lua [Fle17]. Leveraging its design closely aligned with x86_64 processors, the current version of eBPF was introduced in version 3.15 of the Linux kernel [Sta14a]. Essentially, eBPF allows user-space code to run in the kernel environment. Programs are written in a restricted subset of the C programming language and can be dynamically loaded and executed without the need to recompile the kernel or reboot the system. In addition, eBPF programs can also be written in Rust [MK14] using the redbpf tool chain [fon18], enabling memory-safe programming [Jun+17]. The use of Rust, with its interesting safety properties, could make it possible to run extensions in the kernel without having to go through the eBPF verifier, which tends to severely restrict the set of programs allowed to run in kernel-space [Jia+23a].

When a user program is loaded into the kernel, it is compiled into eBPF bytecode, which serves as a low-level representation of the program instructions. The eBPF virtual machine consists of several components, including

a verifier, a just-in-time (JIT) compiler and an execution environment. The verifier ensures the security and correctness of eBPF programs by performing a static analysis of the bytecode, checking for invalid instructions, memory access and security violations. Once verified, the JIT compiler translates the eBPF bytecode into optimized, platform-specific native machine code. Finally, the eBPF program runs in the runtime environment, which provides access to kernel data structures and functions via predefined helpers. Example of these helpers for the Linux Kernel can be found on the Linux User's manual [Lin22]. These helpers enable the program to interact with various kernel subsystems, such as the network, file systems and process management. In addition, the eBPF program can generate events such as tracepoints or user-defined probes, (e.g., eBPF raw tracepoints [Sta18] with eBPF BTF (BPF Type Format) [KaF18] which enables the portability of eBPF programs) facilitating the monitoring and analysis of kernel behavior.

eBPF is a useful tool for efficiently monitoring and manipulating kernel events, including system calls, network packets and file operations [Cas+20; Jia+23b; Zho+22; Zho+23]. It is also used for debugging, profiling and security analysis [DSM19]. In addition, eBPF programs can implement advanced routing mechanisms, improve packet processing speed and enforce security policies [Jad+22; Bon+22; Ber+18a]. The sandbox environment in which eBPF programs operate ensures their isolation from the rest of the kernel, preventing security failures, vulnerabilities or crashes. This characteristic makes eBPF suitable for a wide range of applications [SAD22].

Although originally designed to run custom code in the Linux kernel, eBPF has been extended to other environments, including user-space programs and other operating systems. Projects such as Cilium [Iso17] and XDP [Høi+18] use eBPF to implement high-performance network filters and bypass the Linux networking stack respectively with user space programs. These programs exploit eBPF to intercept and modify network traffic without requiring privileged access to the kernel. In addition, the eBPF ecosystem provides libraries and tools that enable developers to write and run eBPF programs on various platforms, such as Windows, macOS and FreeBSD. Furthermore, due to its GPL license, the eBPF VM has been extracted from the Linux kernel and made available as a user-space library called uBPF [IO 18]. This makes it possible to integrate the eBPF VM into user-space routing protocol implementations.

The introduction of the uBPF virtual machine opens up new possibilities for using eBPF technology beyond the Linux kernel. As an example, uBPF is also the VM used by Microsoft to obtain eBPF support in the Windows kernel [Micb]. Moreover, by integrating uBPF into user-space routing protocol implementations, developers can exploit the flexibility and power of the eBPF virtual machine to incorporate eBPF programs into routing protocols. This enables new approaches that will be discussed in more detail in this chapter

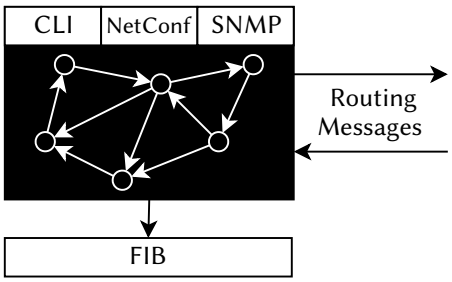


Figure 3.1: Current routing protocol implementations.

and in Chapter 4.

3.2 Pluginizing a Routing Protocol

In this section, we propose a new technique to extend and enhance routing protocols and their implementations. From a high-level viewpoint, an implementation of a routing protocol can be represented as in Figure 3.1. The implementation is modeled as a Finite State Machine (FSM) that exchanges routing messages with other routers. The RFCs describe in detail how and when protocol implementations should send and react to specific packets. This FSM can be configured through the command line interface, SNMP MIBs or NetConf and compute routing tables that are pushed in the FIB. This representation can be seen as a blackbox model, since the network operator is limited to configuring the routing protocol using the interfaces provided by the network vendor. In general, the network operating system is closed and cannot be modified, which means that operators cannot change the FSM. Hence, with this blackbox model, any extension of the protocol requires a replacement of the FSM by the network vendor.

We envision a different implementation model. From a high-level viewpoint, our model is represented in Figure 3.2. We introduce three main modifications compared to the blackbox model. First, the protocol implementation provides a simple API that contains a set of functions that expose the protocol state. For example, an OSPF implementation typically includes functions to add or remove LSAs from the link state database, a BGP implementation includes functions to parse and encode BGP messages. Our second modification is that we allow the FSM that implements the protocol to be extended by adding one or more states, adding one or more transitions or replacing existing transitions. Figure 3.2 shows the FSM that enables the core part of the routing protocol in black and two extensions in red and blue. Our third modification is that we introduce **plugins**. A plugin is some executable code which can be executed inside a routing protocol implementation. A plugin

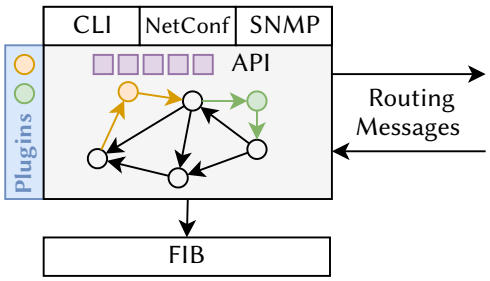


Figure 3.2: Our proposed routing protocol implementations can be extended by using plugins that modify the FSM and use the API.

can use the functions provided by the protocol API and extend the FSM. These plugins enable network operators to design their own extensions to routing protocols and deploy them in their networks without having to wait for their standardization and adoption by multiple router vendors.

In SDN networks, operators can implement new services as software running on a centralized controller that interacts with the network devices through the Openflow protocol [McK+08]. SDN controllers support different programming languages and a range of services have been implemented on them [Kre+14]. A network operator who wants to deploy a new service as a plugin would like to implement it once and deploy it on all routers inside her network.

Our deployment model has several important consequences on the implementation of our plugins. First, it must be possible to execute a plugin on different types of routers that use different CPU models. This implies that either a plugin will be written using a programming language which can be interpreted by the protocol implementation or that it will be compiled into bytecode which is supported by a Virtual Machine that is included in the protocol implementation. Second, OSPF and BGP daemons are always active and it should be possible to extend them without restarting them. Third, since a plugin runs inside the OSPF/BGP daemon, there is a risk that an incorrect plugin could jeopardize the protocol state or even crash it. To cope with these three requirements, we compile the plugins into eBPF bytecode that is executed by a virtual machine that we include in the protocol implementation.

We provide more details on how we extended one implementation of BGP and OSPF to support plugins in the next sections. We first describe the key points of our solution in Section 3.2.1. We detail the management of the memory in Section 3.2.3. We then provide the details related to the OSPF and BGP daemons in Sections 3.2.4 and 3.2.5.

3.2.1 Pluginizing FRRouting

To demonstrate the feasibility of this approach, we apply it to the OSPF and BGP daemons of FRRouting[1]. FRRouting (FRR) is an IP routing protocol suite for Linux and Unix platforms which includes protocol daemons for BGP, IS-IS, LDP, OSPF, PIM, and RIP. It was forked from Quagga and is actively maintained. We used FRRouting version 6.

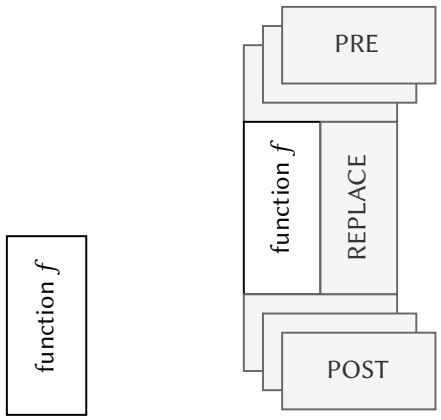
To alter the behavior of both OSPF and BGP, we rely on a user-space implementation of the eBPF [Fle17] virtual machine called uBPF [IO 18] that we linked to the FRRouting daemons. The main advantage of this virtual machine is that it supports the same bytecode as the eBPF virtual machine that is included in the Linux kernel, by definition. It can thus benefit from the different tools that have been written to compile bytecode for the Linux kernel. The uBPF VM can load executable eBPF bytecodes and either interpret them or compile them to x86_64 assembly with its own JIT compiler. Like the eBPF VM of the Linux kernel, it includes a verifier that checks the validity of the loaded bytecode. The uBPF verifier checks (1) all instructions are valid opcodes, (2) there is an `exit` instruction, (3) there is no forbidden operations such as division by zero, writes to read-only registers or invalid jumps, and (4) the memory accesses remain either in its stack or a provided memory area.

Now that the routing daemon includes the uBPF virtual machine, we need to discuss how the daemon must be restructured to enable it to be extended by using plugins. An implementation is organized as a series of functions that process and send packets as well as compute routing tables. These functions are the concrete implementations of specific states of the FSM protocol in a programming language such as C. Henceforth, to enable the modification of the protocol, we use these functions and make them **pluginizable**. These serve as **insertion points** where a network operator can decide to attach plugins compiled in eBPF.

More precisely, one plugin is associated with one routing function and is subdivided in three different parts, called **anchors**, offering a fine granularity on the code injection location. Consider the original function f. The anchors are illustrated in Figure 3.3 and are defined as follows.

- **PRE**: the eBPF code is executed just before running the body of the function f. This anchor can for example be used to load required data inside the plugin or for monitoring purposes. For example, the PRE anchor can monitor the FSM state transition to track the progress of the protocol. Any number of bytecodes can be attached in this mode. They are then executed in a non-deterministic order, but they always terminate before the actual call of function f. Bytecodes attached at PRE

[1] See `https://frrouting.org`



(a) Classical FRR function. **(b) Insertion Point.**

Figure 3.3: Insertion points for eBPF Plugins inside routing functions.

anchors only have read-only access to the routing daemon variables. If no eBPF code is present, this insertion point resumes to a no-op.

- **REPLACE**: the eBPF bytecode is executed instead of the original code of the function f. Only one bytecode can be attached in this mode, and the absence of injected code reduces the REPLACE mode to the original implementation of the function f. This implies that a network operator can dynamically replace one of the functions of the underlying implementation in deployed routers. Bytecodes in REPLACE anchors have read and write accesses to the routing daemon variables. This enables plugins to change key protocol algorithms such as the shortest path computation in the OSPF protocol by including custom network metrics. REPLACE can also be used to suggest new protocol features by redefining the definition of BGP import and export filters. BGP implementations typically propose a domain-specific language (DSL) to design filters, but lack flexibility when designing complex filters. Traditional routing protocol implementations cannot rely on other types of information that the DSL proposes. eBPF can overcome this limitation since it has access to the arguments of the protocol function containing more information than a DSL might propose.

- **POST**: this mode is similar to the PRE one, except that the eBPF code is executed just after running the body of the function f, just before returning to the function it was called from. Bytecodes attached at POST anchors only have read-only access to the routing daemon variables. As PRE function, the POST anchor can be used to monitor the time taken by protocol functions such as compute time of the Dijkstra algorithm in

OSPF. In this case, both PRE and POST are required to track the time when the function starts and ends.

Each pluginizable function has a name that uniquely identifies it. Such convention allows network administrators to easily attach and remove their eBPF scripts in a key-value data structure. Furthermore, such human-readable identifiers provide a convenient interface to dynamically change the plugins attached to the routing daemon without rebooting it. However, this latter method is effectively too restrictive, because routing sessions and routing tables must then be recomputed from scratch. Furthermore, reconfiguration can lead to unauthorized and unnecessary traffic shifts and, in some cases, can even cause instabilities within the network. The literature has shown that network reconfiguration is a computationally difficult problem, falling into the NP-Complete or NP-Hard category, depending on the specific constraints within the network and the routing protocol chosen [SSV22]. A tool such as Snowcap [SBV21] or Chameleon [Sch+23] can be used and adapted for plugins to enable their incremental installation on network routers with minimal impact. The problem of network reconfiguration is in itself a large topic, which is beyond the scope of this thesis.

An eBPF plugin is composed of one or more bytecodes that are attached to a specific insertion point. These bytecodes are called **pluglets** and a plugin can thus contain several pluglets. A given implementation might expose many pluginizable functions. A plugin is defined in a description file listing each ELF file containing the eBPF bytecode and its corresponding function with its insertion point. The network administrator can load it through a command line interface (CLI). Several plugins can co-exist within a routing daemon. To be executed, pluglets require a VM. The current uBPF implementation provides an API to create a VM containing the loaded eBPF bytecode to execute it. However, such VM can only contain one bytecode at a time. Furthermore, with uBPF there is no API to update the code attached to a VM. The code replacement is nevertheless a required feature to dynamically update plugins, which is important for routing protocol daemons that never terminate. To solve these problems, we extend the API to manage multiple plugins. In fact, we create a specific instance of uBPF machine which is in charge of only one pluglet. Several of these VMs can be attached to a routing daemon at a given time. These multiple VMs are stored into a map, each being associated to a plugin. This map is, of course, accessible through our extended uBPF API inside the routing daemon.

3.2.2 Executing a Plugin Inside the eBPF VM

The VM has two modes of executing eBPF bytecode: interpreted and JIT (Just-In-Time) compilation. In the interpreted mode, the VM reads and executes

each eBPF assembly instruction without converting it into machine language beforehand. Instead, the eBPF program is executed within another program integrated in the routing daemon. To achieve this, it emulates an "eBPF processor" to execute the program, keeping the eBPF code isolated from the rest of the daemon. Even if the eBPF code fails, it will not affect its execution.

The JIT compilation mode directly translates eBPF instructions into machine code understood by the CPU. Instead of interpreting the instructions, the VM converts them into assembly code that can be directly executed by the CPU. This mode is faster because there is no intermediary step, but it can be risky as the code is directly executed by the CPU. The eBPF program can potentially cause issues like accessing unauthorized memory or crashing the process.

Typically a JIT compiler defers the compilation of eBPF bytecode until it can generate optimized machine code that takes into account the program's current execution. Machine code is generated while the eBPF program is running. However, the JIT compiler integrated in the uBPF VM is basic and can only compile the entire eBPF bytecode before execution.

3.2.3 Memory Management

One of the motivations for using VMs is their isolation from the routing daemon they are attached to. In addition, the eBPF instruction set is quite small and simple, making it easier to control their operations. However, plugins may require more information from the implementation than the initial arguments provided to the VM. To exchange information with the routing daemon, FRRouting registers a set of functions that are made accessible to the uBPF VM, and therefore the plugins. As both pluglets and FRRouting are written in C language, plugins could theoretically access any memory location within the routing daemon. In practice, this could create stability problems if badly written eBPF code tries to access invalid memory locations. Furthermore, it would make plugins very dependent on the FRRouting internals that may change over time. To ensure the stability of the executable that combines the routing daemon and the eBPF plugins, we leverage the uBPF VM to control the memory that a given plugin can access. This is done through different techniques.

First, the routing daemon exposes through an API a set of getter and setter functions to access the main data structures (packets, LSDB for OSPF, RIB for BGP, etc.) maintained by the routing daemon. These functions are part of the modified routing daemon. They also verify the validity of their input parameters.

Second, the different pluglets composing a given plugin may need to collaborate together by exchanging information. Each pluglet is supported

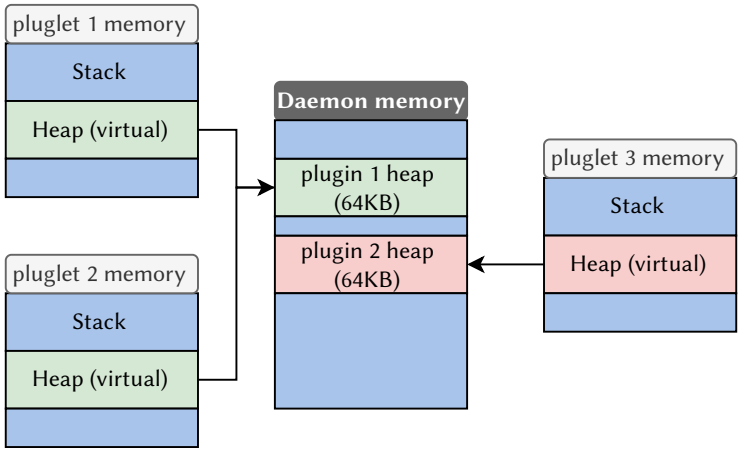


Figure 3.4: The two pluglets of the left plugin share the same heap while the pluglet of the right plugin uses a separate heap.

by one instance of the uBPF VM and has its own stack. To address this requirement, FRRouting keeps a dedicated **context** for each plugin. Thanks to this context, we can associate a plugin-specific heap that is shared among the different pluglets that compose a plugin. These pluglets can allocate and free memory in their shared heap by using functions that are similar to `malloc(3)` and `free(3)`. This is illustrated in Figure 3.4. As we want to keep control on the memory used by the plugins, we do not directly expose the associated functions of the C library. Rather, we reimplement some of them like `memcpy(3)`, `malloc(3)` and `free(3)` and expose them to the uBPF VMs. In addition, the API of the routing daemon provides functions for pluglets to map an area of the plugin heap to a plugin-specific identifier. Such mechanisms enable collaborative pluglets to retrieve a precise memory area while providing isolation between plugins.

3.2.4 Pluginizing the OSPF Daemon

The previous sections described the generic techniques that are required to add plugins to a routing daemon. In addition, the routing daemon also needs to expose specific OSPF functions that the plugins can use. We briefly describe these OSPF functions and the insertion points in this section.

The insertion points of an OSPF daemon are the protocol functions where eBPF plugins can be attached. These insertion points depend on the features that eBPF plugins need to support. Our prototype includes several insertion points. We briefly describe some of them. The `SPF_CALC` insertion point corresponds to the function that computes the shortest paths. The `OSPF_SPF_NEXT` insertion point corresponds to a function which is part of the SPF calculation

process that implements Section 16.1 of the OSPF specification [Moy98]. The `HELLO_SEND` insertion point corresponds to the function that sends Hello packets. The `LSA_FLOOD` insertion point corresponds to the function that floods the received LSAs. The `ISM_CHANGE_STATE` insertion point corresponds to the function that is called when an interface changes the state of its Interface State Machine.

The OSPF API also exposes some functions to the plugins. First, we expose functions used to get/set some OSPF internal structures. For example, the `get_ospf_area` function is used to get a copy of an OSPF area structure from OSPF while `set_ospf_area` can be used to set an OSPF area structure to a desired value. Such functions are provided for most of the important structure maintained by OSPF. We also expose functions from the implementation that can be useful for plugins. Examples of such functions are `plugin_ospf_flood_through_area` that allows flooding an LSA through an area and `plugin_ospf_lsa_install` that allows installing an LSA in the router's LSDB.

3.2.5 Pluginizing the BGP Daemon

The BGP daemon is also extended similarly. We add insertion points on functions receiving BGP messages from neighbors, on filters and inside the decision process. We also expose specific functions to the plugins that are executed by the uBPF VM.

Our BGP API exposes two types of functions to the eBPF plugins. First, there are functions to access/modify some elements of the data structures maintained by the BGP daemon. For example, `get_cmp_prefixes` is used to retrieve two prefixes received during the BGP decision process. The first one is a prefix received from the remote peer when it has sent a BGP Update message. The second prefix is one prefix already present in the Adj-Loc-RIB. The `get_attr_from_prefix` returns the attribute structure related to the prefix sent by a remote peer. The `as_path_from_prefix` function returns the AS path related to a prefix while the `get_attr_from_path_info` returns all the attributes of the update passed as argument. The `get_community_from_path_info` extracts the BGP community structure associated with a given path.

Our BGP API also includes functions that manipulate and compare BGP messages or their attributes. These functions are typically used by the BGP decision process and will be used for one of our use cases. Example functions include `aspath_cmp` that compares two AS paths (i.e. their length and the ASes they contain), `aspath_count_hops` that returns the number of ASes contained in a given path, similar functions for the MED or other BGP attributes or the `peer_sort` which determines whether a peer is an eBGP or an iBGP neighbor.

3.3 Use Cases

In this section, we describe four examples showing how network operators and researchers can leverage the proposed plugins to extend a routing protocol.

First, we demonstrate in Section 3.3.1 that we can use plugins to extract and expose internal protocol information for monitoring. Second, we show in Section 3.3.2 plugins changing the protocol packet format and its interpretation in the OSPF route computation. Third, we show in Section 3.3.3 plugins describing more expressive BGP filters. Fourth, we demonstrate in Section 3.3.4 that plugins can also modify the BGP decision process.

3.3.1 Monitoring routing protocols

One of the most popular use cases for eBPF in the Linux kernel is to monitor various events that occur inside the kernel in an efficient and non-intrusive manner. Similarly, we added monitoring facilities to the BGP and OSPF implementations in FRRouting.

To illustrate the monitoring capabilities of our proposed plugins, we have designed and implemented both a BGP and an OSPF monitoring daemons that interact with plugins running on the routing daemons and export statistics using IPFIX [ACT13]. Those statistics are aggregated by the daemons and exported to an IPFIX collector.

To monitor the BGP routing daemon, we implemented several BGP plugins that are attached at PRE anchors at several insertion points. Some of these plugins monitor specific BGP messages. For example, our plugin monitoring the Open messages, used to start a BGP session, is composed of 50 lines of C code and uses 16 API helper calls. Similar plugins are provided for the Keepalive and Update messages. Besides monitoring the received BGP messages, one plugin also measures the time required to run the BGP decision process. We have also implemented plugins that track specific IP prefixes or analyze the received AS Paths to enable the operator to provide more detailed statistics. Finally, we built plugins in charge of both monitoring withdrawn and rejected routes. The last one provides the reason of the rejection decided by BGP import filters. We study the performance impact of these plugins in Section 3.3.3.

We also implemented similar plugins to monitor OSPF. These plugins are inserted at the PRE and POST anchors of different insertion points. With plugins shorter than 10 lines of code, we can monitor things such as the execution time of the SPF calculation process, the number of Hello packets sent or the LSAs flooded by a router.

3.3.2 More flexible OSPF route computation

One of the benefits of our proposed plugins is that it is possible to extend the routing protocol. As an illustration, we implement a new type of OSPF LSA and update the shortest path computation algorithm. This idea is similar to the flexible IGP algorithm standardized by the IETF [Pse+23] in February 2023. We do not adopt the syntax proposed by the IETF standard, but the idea is similar.

We first define a new OSPF LSA (type 13). This LSA is similar to the normal router LSA (type 1), except that we associate an additional metric (as an integer) to each link. We use this additional metric to represent the color of each link. This plugin is implemented by using about 100 lines of code and is inserted in the `SPF_CALC` insertion points. This LSA is then flooded inside the network.

Our second plugin is attached at the REPLACE anchor of the `OSPF_SPF_NEXT` insertion point. It modifies the `ospf_spf_next()` function which implements Section 16.1 of the OSPF specification [Moy98]. In this function, the LSDB is represented as a directed graph. The `ospf_spf_next()` function examines the links in the LSAs of the first vertex from the candidates' list. Then it updates the list of candidates with any vertices that are not already on the list. If a lower-cost path is found to a vertex already present in the candidate list, it stores the new cost. We rewrite this function as an eBPF plugin to support color constraints. For each router LSA that is examined, we check if there is a corresponding (same router-ID) LSA of type 13 in the LSDB. If yes, we check for each link the color of the link. If it is green, we continue normally, if it is red, we ignore the link. This plugin is implemented in about 160 lines of C code.

As an illustration of the utilization of this new LSA, we simulated the network topology shown in Figure 3.5. In this network, all the links have a cost of 100. With the standard Dijkstra algorithm, router `R1` uses its direct link to reach both `R2` and `R3`. With our type 13 LSA, `R1` advertises a different color (red) for the directed link between itself and `R3`. This LSA is flooded in the network and our second plugin running on the different routers computes the routing tables without considering the red link. In this configuration, `R1`'s routing table forwards packets destined to `R3` via `R2`. On the other hand, `R3` continues to use its direct link to reach `R1`.

To evaluate the performance of these OSPF plugins, we loaded the OSPF router with the topology of the GTS Central Europe network from the Internet Topology Zoo [Kni+11]. It contains about 150 nodes and the same number of edges. This is one of the largest topologies from this public dataset which makes it a good candidate to evaluate this plugin overhead.

We used this experimental setup to evaluate the memory and CPU con-

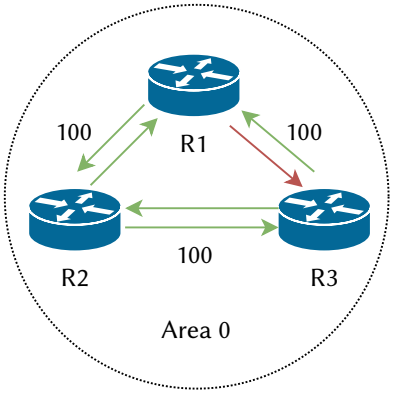


Figure 3.5: Simple OSPF network.

sumption of our plugins. For this, we consider three different versions of the OSPF daemon: (*i*) the **vanilla OSPF** daemon from FRRouting version 6, (*ii*) our **flexible OSPF daemon** but without any plugin and (*iii*) our **flexible OSPF daemon with two plugins** installed (the two plugins that allow changing the Dijkstra computation using our new type of LSA). Looking at the memory consumption, we observe that without plugins, our **flexible OSPF daemon** consumes 4.93 MBytes of memory while the **vanilla** one only consumes 4.85 MBytes. This difference is due to the additional data structures required to support the management of the plugins. With the two plugins loaded, the memory consumption grows to 5.23 MBytes. The difference between the flexible OSPF without plugins and the one with plugins is due to several reasons. First, a 64 KBytes heap is dedicated to each plugin when it starts. This heap remains allocated for the future executions of the plugin. Second, the bytecodes of the plugins consume between 1 and 10 KBytes per plugin. Third, when bytecodes are injected, the implementation stores some more metadata related to it and the uBPF VM also maintains data related to the VM state. All this together leads to about 300 KBytes of overhead for two plugins. This seems reasonable for today's routers.

To evaluate the CPU cost of the plugins, we focus on the computation of the shortest paths, which is the most important algorithm used by an OSPF daemon. To measure the CPU time required to compute the shortest path, we rely on the following experiment. We start the router under test, let it download the entire LSDB from its neighbor and measure the time required to compute the shortest path after the full transfer of the LSDB. Figure 3.6 provides the CPU times measured over 50 different runs with four variants of our flexible OSPF daemon. Our baseline is the **vanilla OSPF** daemon which takes on average 5.34 msec to compute the shortest paths. Our flexible OSPF daemon takes roughly 5.49 msec without plugins and 5.51 msec with the

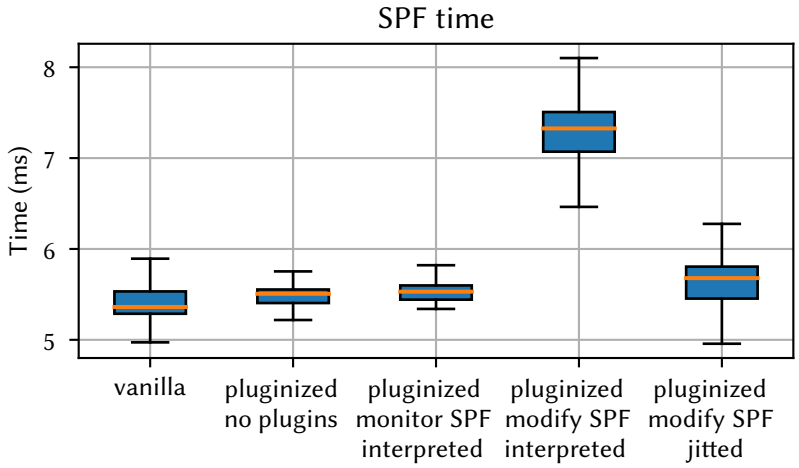


Figure 3.6: SPF execution times over 50 runs on the emulated 150 nodes GTS Central Europe topology.

monitoring plugins.

The eBPF plugins can either be interpreted or compiled by the JIT compiler of the uBPF VM. The plugin that supports our Type 13 LSA inside the shortest path computation is composed of about 160 lines of C code. It is executed for each node/edge visited during the shortest path computation. When this plugin is interpreted, the shortest path computation increases up to 7.3 msec. However, once the plugin is compiled by the JIT, the shortest path computation time drops to 5.66 msec and thus the overhead remains small compared to vanilla OSPF.

3.3.3 More flexible BGP filters

FRRouting, like most BGP implementations, supports a range of import and export filters. A network operator can write access-lists that define the list of prefixes which are accepted/rejected. It is also possible to specify a prefix-list which can also match on the prefix length. FRRouting also supports filters that match on the AS-Path and route-maps which can match on other attributes such as BGP Communities, the origin of a route, the peer that announced a route. Such filters are widely used by network operators [FB05] and some router configurations contain thousands or tens of thousands of lines of configuration files to specify them.

Although route-maps are the most flexible BGP filters, their configuration might become cumbersome and complex [FB05]. Our proposed eBPF plugins enable network operators to write filters in C code which is much more expressive than the ad-hoc languages that have been defined by router vendors

to support filters. Furthermore, such eBPF filters could access to additional information, such as the current state of the protocol.

The filtering process is supported per peer and per prefix and defined in a single function inside FRRouting. We added an insertion point for the eBPF virtual machine inside this function. However, there are situations where an operator could want to attach several eBPF plugins to this filtering process. Given that the order of the application of the filter functions can be important for the decision of the filter, we allow the network operator to specify the order in which different REPLACE plugins will be executed for this filtering function.

An eBPF plugin for the filtering function can return three different results: `FILTER_DENY` if the filter has decided to reject the route, `FILTER_ACCEPT` if the filter has decided to accept the route and `BGP_CONTINUE` if the next filter needs to be applied. The uBPF virtual machine executes the different BGP plugins in the order specified by the network operator when it loaded them and stops the processing as soon as one of them returns `FILTER_DENY` or `FILTER_ACCEPT`.

We previously mentioned that a filter could modify protocol variables. As for traditional filters, the virtual machine enables an eBPF filter to modify attributes such as the local-preference, the MED, the AS-PATH (for path prepending for example), BGP communities, etc. The eBPF filters can also read the current RIB of the BGP router. This could bring new filter possibilities based on the RIB content like inserting a prefix if another is present or missing.

Another advantage of eBPF filters is that it becomes easy to manipulate BGP communities. Many network operators use BGP communities for a wide range of purposes [DB08]. Measurements indicate that BGP routers rarely remove the BGP communities that upstream routers attached. This increases the size of the BGP routing tables and the memory consumption on routers and opens a range of operational problems [Str+18]. With eBPF, the programmer

Table 3.1: Data about eBPF plugins for the decision process.

eBPF Function	API Calls	eBPF Insts	LoC
Local Pref	21	85	51
As Path	19	91	52
MED Check	25	147	58
IGP Weight	76	338	134
Router ID cmp	26	110	54

defines which communities to match without maintaining complex route-maps that are sometimes hard to read and difficult to manage. An eBPF filter can easily add, delete and compare BGP Communities since the filter is written in C and has more flexibility than the router Domain-Specific language (DSL).

We developed several BGP import filters as eBPF plugins to illustrate the flexibility of our proposed architecture and evaluate the performance impact of these new filters. Our first plugin is used as an import filter. It simply parses the AS-Path as only accepts the routes advertised by an odd-numbered AS. This filter only requires 5 lines of C code. We do not expect that network operators would want to use it in their network, but we use it as a simple benchmark to evaluate the performance impact of the eBPF plugins.

Our second eBPF plugin is more useful for network operators. The BGP routing tables in the default-free Internet continue to grow. Recent data shows that routers of Route-View project need to carry more than 1.2M routes [Hus23a]. Recent routers can easily handle such large routing tables, but many smaller ISPs and enterprise networks still use older routers that have limited memory. On such routers, it makes sense to only accept a subset of the routes to avoid overflowing the available memory. Many ISPs use filters to block IPv4 prefixes that are too long (e.g., `/24`) [Upa; Cit+10]. However, these filters block some legitimate prefixes. Measurement studies have shown that a small fraction of the ASes that advertise prefixes are responsible for the pollution of the BGP routing tables by advertising many more specific prefixes that are covered by a less-specific one [Cit+10]. Some ASes advertise both a `/20` IPv4 prefix and all the `/24` subprefix that it contains. Our second eBPF plugin automatically detects those ASes that de-aggregate their large prefix and only accepts the first 4 more specific prefixes that are included inside a larger one that is already included in the router's RIB. This eBPF plugin is implemented using 19 lines of C code. When a BGP route is received, the eBPF plugin verifies whether it is already covered by a less specific prefix that already includes 4 more specific prefixes. If so, the route is rejected, otherwise it is accepted. Of course, this plugin can introduce BGP routing problems, such as routing loops, particularly if the initial selection of the first four prefixes by all routers on the network differs. Even if the four prefixes are identical, as routers converge in an unpredictable order, the network can be unstable during the convergence. Solving this convergence problem is beyond the scope of this thesis.

To evaluate the performance of these two filters, we consider the simple scenario shown in Figure 3.7. Router `R1` uses `exabgp` to inject a BGP routing table [2] containing 200K entries to router `R` over an eBGP session. Router `R` uses different versions of FRRouting. The host running router `R` is equipped

[2] For this evaluation, we rely on the BGP routing table from Spotify's `super-smash-brogp` project, see `https://github.com/spotify/super-smash-brogp`.

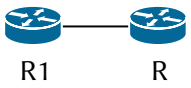


Figure 3.7: Network topology used to evaluate the performance of the BGP filters implemented as eBPF plugins.

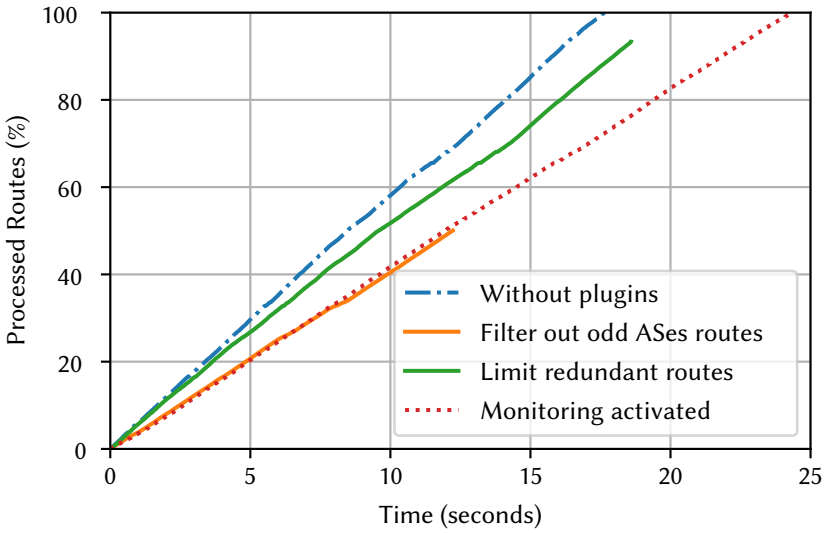


Figure 3.8: Performance of the BGP filters implemented as eBPF plugins and executed in interpreted mode.

with an Intel(R) Core(TM) i3 CPU 540 @ 3.07GHz running Linux kernel 5.0.13, 12GB of RAM and one 1 Gbps NIC.

Our baseline for the evaluation of the performance impact of the eBPF filters is the utilization of FRRouting without any filter. We add to FRRouting an eBPF plugin that monitors the insertion time of each prefix in the router FIB. Using this plugin, we plot on Figure 3.8 (dotted blue curve) the time required to process the BGP updates received from R1. The green curve shows the time required to process the same BGP updates with our second eBPF plugin that filters the more specific prefixes. This filter rejects 13k of the 200k routes and only increases the processing time by 5.4%. The eBPF filter that rejects the routes advertised by an odd-numbered AS processes all the BGP routes in only 12.23 seconds, but it rejects half of them. Since it rejects many routes, the BGP daemon has to perform less computation than when there is no plugin.

The dotted magenta curve on Figure 3.8 shows the time required to process all the BGP updates sent by R1 without any filter but with the 7 eBPF monitoring plugins described in Section 3.3.1 installed.

3.3.4 Pluginizing the BGP Decision Process

Our last use case is the BGP decision process. This is a key part of the BGP daemon that controls the selection of the best path towards each destination prefix. Network operators use various techniques to influence the selection of these best paths [FBR03; Quo+03; Tei+04]. Some routers can be configured to skip some steps of the BGP decision process or slightly modify their behavior [FR07]. For example, many BGP implementations support a configuration parameter to always compare the MED attribute even between routes that were received from different peers.

In the FRRouting BGP daemon, the decision process is implemented as a single function (`bgp_path_info_cmp`). We refactor the FRRouting code to organize this function such that it now calls one specific function per step of the BGP decision process. Each of these steps is then implemented as a separate function. These functions are all implemented following the same pattern. They compare a new path with the best one that is already present in the BGP routing table. If the new path is strictly better based on the attributes that are compared in this step of the decision process, then the function returns `BGP_COMP_SPEC_2`. If the best current path is strictly better than the new one, the function returns `BGP_COMP_SPEC_1`. If the two paths are equivalent according to the attributes considered in this step of the decision process, then the return value of the function returns the next step that needs to be executed. This makes it possible to fully customize the BGP decision process, not only replacing one step with another, but changing the order of the rules of the BGP decision process.

Thanks to the utilization of eBPF plugins, network operators can easily tune the BGP decision process of their routers. Many network operators use BGP communities to tag the Point of Presence (PoP) or the city where a given route was learned [DB08]. We leverage this to implement a variation on hot-potato routing that uses the geographical distance between PoPs to prefer one route over the other. Each PoP is encoded as a BGP community and our eBPF plugin contains a table with the latitudes and longitudes of all the PoPs of the ISP. When two routes are compared, the eBPF plugin computes the distance between them based on the geographical coordinates of the PoPs where they were received and always prefers the closest one. This eBPF plugin is implemented in 148 lines of C code.

To evaluate the cost of using eBPF plugins within the BGP decision process, we reimplement all the steps of the decision process as plugins. In FRRouting version 6, there are 14 different steps in the decision process. The eBPF plugin that supports the `local-pref` attribute requires 51 lines of code. This is one of the simplest steps of the BGP decision process. The most complex eBPF plugin is the one that compares the IGP cost towards the BGP nexthop. This

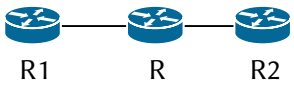


Figure 3.9: Network lab used to evaluate the pluginized BGP decision process.

eBPF plugin is implemented using 134 lines of C code. We do not expect that network operators will replace all the steps of the BGP decision process with eBPF plugins, but use this as our worst-case scenario to evaluate the performance penalty of these eBPF plugins.

We consider the network shown in Figure 3.9. Router R1 sends a full BGP routing table containing 200k routes. Once router R has accepted all the routes announced by R1, router R2 starts to announce exactly the same routes. Every route sent by R2 must be evaluated by all the steps of the BGP decision process on router R that eventually prefers the new one because of its `router-id`, i.e. the last step of the BGP decision process.

We consider different variants of our modified version of FRRouting and measure the execution time of the BGP decision process to fully process routes sent by R2. As for evaluating filter performances in Section 3.3.3, we use the same eBPF plugin that monitors the insertion time of each re-advertised prefix in the router FIB. With the vanilla BGP daemon, the dotted blue curve of Figure 3.8 shows that it takes on average 17.5 seconds to accept all the routes sent by router R2 and install them in the FIB of router R. If we use our modified version of FRRouting that supports eBPF plugins but do not install any of them, router R needs up to 28 seconds to install all the routes sent by R2 in its FIB. Finally, when all the steps of the BGP decision process are implemented as plugins, router R needs almost 34.8 seconds to install the same number of routes in its FIB. We instrumented our code to analyze the reason for this high cost of the pluginized BGP decision process and have identified that the simple linked-list used by the memory allocator of our prototype was the culprit. In the upcoming chapter (Chapter 4), a more efficient version of the plugin memory allocation is proposed to address this problem.

3.4 Related Work

Based on feedback from their customers, router vendors have implemented various techniques to control the operation of routing protocols. The Command Line Interface (CLI) is the classical way for network operators to tune the configuration of the routing protocols running on their routers. Some also rely on SNMP MIBs to gather statistics and some simple configuration tasks [HH06; Joy+06]. Over the years, router vendors have added new techniques to enable their customers to interact with the router software. Some vendors provide

scripting facilities [LS16; Cis11] and the industry is now heading towards the utilization of Yang models [CCL19]. However, these approaches do not enable network operators or researchers to extend the underlying protocols.

The eBPF virtual machine has been introduced in the Linux kernel a few years ago. It is now mainly used for configuration and monitoring purposes [FC19]. Looking at the networking use cases, eBPF is used to provide fast programmable data packet processing [Gre+05], improve firewalls [Ber+18b], implement network services [Mia+18], support IPv6 extensions [XDB18], extend TCP [TB19] or implement Multipath TCP schedulers [Frö+17]. We are not aware of applications of eBPF to routing protocols.

In the late nineties active networks were proposed as a solution to bring innovation back inside the network that was perceived as being ossified [TW96]. Most of the work in this area focused on the possibility of placing bytecode inside network layer packets. This bytecode was then executed by virtual machines running on routers. The idea of placing code inside packets was not adopted by industry [Cal06], but P4 [Bos+14] could be considered as a modern variant of this idea. In the control plane, researchers built upon this idea to propose new solutions such as the 4D architecture [Gre+05], the Routing Control Platform that centralizes routing [Cae+05] or Metarouting [GS05] that proposed to open the definition of routing protocols using a declarative language.

Although the eBPF plugins proposed in this chapter were applied to BGP and OSPF, the same technique could be used with other control plane protocols. There are several ongoing efforts to develop new routing protocols that could benefit from such plugins. Some examples include Facebook's Open/R routing platform [Has+] or the protocols that are being designed within the LSVR [Pat+23], RIFT [Prz+23] or BABEL [CS21] IETF working groups.

3.5 Conclusion

In this chapter, we have presented a way for network operators to improve their routing protocols without depending on router vendors. To achieve this, we have integrated a lightweight virtual machine (VM) that uses the eBPF instruction set with a customized runtime environment dedicated for the routing protocols. By integrating this VM with BGP and OSPF, operators can dynamically run their own plugins without having to reboot or recompile the routing protocol implementation. This means they can easily extend and update the router with new features, or make modifications to existing ones. To demonstrate the feasibility of this approach, we first integrated the modified eBPF VM in FRRouting, an open-source implementation of routing protocols and second, we have implemented different use cases in both OSPF and BGP. For OSPF, we have developed advanced monitoring capabilities and a more

flexible and efficient route calculation method by "coloring" links in LSAs. On the BGP side, we have shown that it is possible to implement complex import/export filters using only a few lines of C code, and that the BGP decision process can be modified.

The solution described in this chapter is specifically designed for FRRouting version 6, which limits its applicability to operator networks using routers from different vendors. In the following chapter, we will develop our solution to ensure compatibility between different platforms, making it platform agnostic.

*x*BGP: Faster Innovation in Routing Protocols

4

This chapter is largely based on the work T. Wirtgen, T. Rousseaux, Q. De Coninck, N. Rybowski, R. Bush, L. Vanbever, A. Legay, and O. Bonaventure. "xBGP: Faster Innovation in Routing Protocols". In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 35–50. ISBN: 978-1-939133-27-4. URL: https://www.usenix.org/conference/nsdi23/presentation/wirtgen. This thesis' author's contributions involves all the work discussed in this chapter. The two first authors contributed equally on the verification part of *x*BGP programs. This chapter will primarily focus on the verification aspects that the thesis' author worked on. Additional aspects related to the verification can be found in the corresponding paper.

In the previous chapter, we discussed the process of extending a routing protocol implementation using eBPF. This was a first step towards realizing our vision where network operators use plugins to extend their distributed routing protocols. However, it is important to note that this method is tailored to a specific implementation of BGP (or OSPF). Consequently, any modification or update to the internal structure of this implementation may cause previously developed plugins incompatible. This limitation is due to the fact that the integration of our eBPF virtual machine relies heavily on existing functions and data structures in this particular implementation.

Furthermore, as mentioned in Chapter 2, networks are generally composed of routers from different vendors, each with their own BGP implementations. Relying on a single implementation is impractical, as it does not take into account the diversity of routers and their distinct protocol implementations. If operators were to adopt the plug-in approach explained in Chapter 3, they would have to create separate plugins for routers from different vendors. This approach is cumbersome and unsuitable for large-scale networks.

Given these considerations, it becomes clear that a more flexible and scalable solution is needed to accommodate the diverse nature of network environments. It is imperative to reevaluate the plugin approach to ensure compatibility between specific implementations, and to take into account

the heterogeneous landscape of router vendors and their respective protocol implementations. By taking a broader perspective, we can develop adaptable solutions that support networks with routers from different vendors without compromising scalability and efficiency.

In this chapter, we argue for a much lighter weight and practical approach to network control plane programmability by *allowing the network operators to easily extend the distributed routing protocols that they already use, no matter what implementation is used on the router.* Our new approach, which we call *x*BGP, is inspired by the success of the extended Berkeley Packet Filter (eBPF) in Linux [Gre19; Gre+05] and Windows [Micb]. In *x*BGP, different BGP implementations expose an API and an in-protocol VM with a custom instruction set to access and modify the intrinsic protocol functions and memory. Thanks to this API and the VM, the *same* code can be executed on different implementations. Note that the instructions set and the in-protocol VM still need to be adopted and implemented by each vendor, but this is a one-time effort, instead of a *per-feature* effort.

Naturally, opening up BGP implementations to external programs opens the door to many (research) questions: *Which API should BGP expose? How to implement this API efficiently* or *What about the correctness and the safety of these extensions?* We answer these questions in this chapter and make four main contributions.

First, we introduce the x*BGP API* which defines a set of functions that expose the key functions and data structures that should be supported by an extensible BGP implementation. As network operators often use network equipment from multiple vendors, it would be ideal if the same bytecode could be executed on different routers. This was not yet possible with the prototype of Chapter 3. In different domains, browser vendors have agreed on supporting the WebAssembly virtual machine in their browsers [Haa+17] and operating systems vendors have agreed on using the POSIX API. We present our API in Section 4.1 and describe how we modified two different BGP implementations, BIRD and FRRouting, to support *x*BGP.

Second, network operators cannot take the risk of running *x*BGP extensions that could jeopardize the correct operation of their routers. In the previous chapter, we saw that operators can inject arbitrary plugins into their BGP or OSPF implementation. However, these plugins run the risk of causing router failures. This risk is considered unacceptable, as routers are expected to generally run continuously without interruption. Network operators, notably in enterprise and ISP networks, usually test new versions of the router operating systems in their labs before deploying them in production networks. These tests are intended to prevent problems when new versions of the software are deployed in production. They typically run for days or weeks and are only used before major upgrades. With protocol plugins, i.e., network stack implementa-

tions that are dynamically extensible through platform-independent bytecode, network operators will operate in a more agile manner. They could change deployed plugins on a daily basis. However, they are unlikely to perform manual tests with such plugins. A better approach would be to use automated tools that leverage formal methods to validate that the plugins are correct. The Linux kernel, which also includes an eBPF virtual machine, solves this problem by strictly limiting the memory area that eBPF programs can access and the number of instructions that they execute. The latter limitation is removed in our VM version to support more complex use cases. Researchers have recently proposed software tools that rely on formal methods to validate such eBPF code [Ger+19]. Pluginized QUIC [De +19], which applies a similar idea to extend the QUIC protocol also uses verification technique to prove than plugins terminate. This is a first step for the validation of plugins. We then present in Section 4.2 a complete validation workflow that enables operators to validate that their extensions correctly terminate, do not interfere with the memory of the host implementation, produce syntactically valid BGP messages, or only use the *x*BGP API functions authorized by the network operator. We envision this workflow to become one element of the qualification tests that operators already carry out before deploying any new BGP feature in their network.

Third, plugin performance should be as close as possible to the performance of native code, but this performance gain should not come at the expense of the safety guarantees. One possible direction to improve performance is to try to extract as much performance as possible from our modified eBPF virtual machine that runs in userspace. One of the performance bottlenecks of the prototype of Chapter 3.2 is the memory accesses that are audited by our virtual machine to prevent out-of-bound memory accesses. These checks consume CPU time and affect performance. One possibility to reduce them would be to develop a plugin compiler that ensures that JITed plugins only access authorized memory areas. Such tools exist for standard executables [Pan+19]. Another approach utilized in *x*BGP, involves the use of our validation workflow verifier. The need for runtime checks is eliminated as memory accesses are verified in advance. We demonstrate the practicality of *x*BGP in Section 4.3 by measuring its overhead compared to native implementations. Even for complex extensions (re-implementing BGP Route Reflection), our benchmarks show that the overhead of *x*BGP is always under 13%, a reasonable value given the flexibility benefits.

Fourth, we showcase the practicality of *x*BGP in Section 4.4 by implementing eleven use cases with *x*BGP in which we extend BGP to: support a new BGP attribute; introduce new selection rules; restrict the set of paths it can compute; detect unused routes (zombies); or monitor BGP operations. Each use case involves the same *x*BGP bytecode running on both FRRouting and BIRD.

4.1 Architecture

At a high level, *x*BGP enables network operators to customize or extend any compatible BGP implementation by injecting and directly executing *x*BGP programs. As an illustration, we consider how to expand a BGP implementation to support a new BGP attribute, *GeoLoc*, that stores the geographic location (i.e., longitude and latitude) where each BGP route was learned. Among others, this attribute can be used to adapt router decisions, e.g., by filtering away routes learned more than x kilometers away. Supporting such an attribute has been discussed within the IETF but never standardized [CSR16]. Yet, large-scale ISPs reportedly use iBGP filters [VCD14] to achieve the same effect. Using iBGP filters is risky though as doing so can lead to permanent oscillations [VCD14].

To implement the *GeoLoc* extension, we need to support several operations in a BGP implementation. (1) When a route is received over an eBGP session, the router adds a new attribute, `Geo_Originator` that contains the geographic coordinates of the router that learns the BGP route in an import filter. (2) If the BGP route already contains the `Geo_Originator` attribute, the router needs to decode it. (3) When exporting the route to another peer, the router can use the `Geo_Originator` attribute to filter routes that are too far away. (4) To be usable by other iBGP peers, the attribute needs to be added to the BGP Update message.

To add this extension, we need to understand how BGP implementations are designed. There are many ways to organize a BGP implementation. Each implementer selects a particular software architecture and the associated data structures based on their own requirements. However, all BGP implementations must adhere to the protocol specification [RHL06]. This specification defines the format of the BGP messages, an abstract BGP Finite State Machine that manages each BGP session, and also an abstract workflow and data structures that describe how BGP update and withdrawal messages should be processed. This workflow is illustrated in black in Figure 4.1.

Starting from the left, a received BGP message is stored in the `Adj-RIB-in`[1]. It then passes through the `import` filters that may decide to discard the message or modify attributes such as `local-pref`. If the route is accepted by the import filters, it is inserted in the `Loc-RIB`. The `Loc-RIB` contains all the BGP routes accepted by the router. The BGP decision process extracts from the `Loc-RIB` the best routes that are placed in the `RIB`. These routes then pass through the `export` filters before being advertised over BGP sessions.

Going back to our *GeoLoc* extension, we can see that it can be added to the different parts of the BGP workflow. (1) needs to be added to the part that

[1] Some implementations do not explicitly maintain a separate `Adj-RIB-{in,out}` to reduce their memory consumption and store everything in the `Loc-RIB`. We ignore this implementation detail in this chapter.

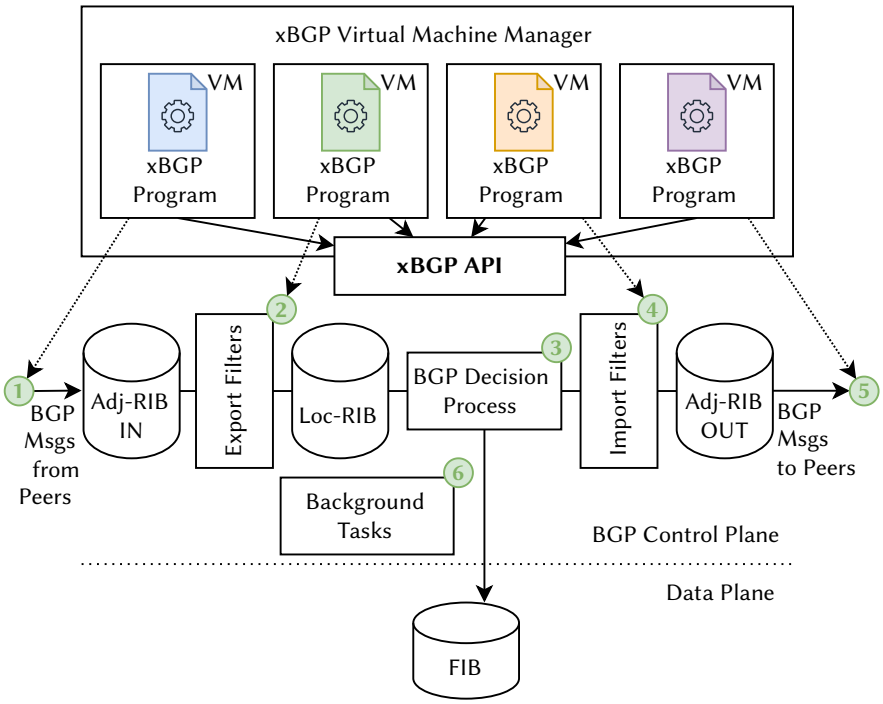


Figure 4.1: An *x*BGP compliant implementation exposes the abstract BGP data structures defined in RFC4271 through a generic API and uses `libxbgp`'s Virtual Machine Manager to attach the bytecode that implements extensions to specific insertion points (green circles).

parses a BGP attribute. (2) and (3) must be designed as import and export filters respectively. And (4) will be added to the serialization part of the BGP implementation. The question now is how to add those subcomponents to the main BGP implementation. To answer this, we defined the insertion points depicted in Figure 4.1 with the green circles on which functionalities can be added or modified. These insertion points correspond to the major BGP events. It is now easy to add the four components of our simple extension to the BGP implementation in their respective insertion points. (1) is attached to the `BGP_RECEIVE_MESSAGE` ① insertion point. First, it queries the BGP neighbor's table and determines the type of the eBGP session. Then, it retrieves the contents of the received BGP update in network byte order. Finally, it attaches the new GeoLoc attribute to the route. The second program (2) is attached to the `BGP_INBOUND_FILTER` ② insertion point. It retrieves the router coordinates from the router configuration to add them to the attributes of the route. The program (3) attached to the `BGP_OUTBOUND_FILTER` ④ retrieves the neighbor information and the *GeoLoc* attribute to check if the route can be advertised to the peer. Finally, the fourth program (4) is attached to the `BGP_ENCODE_`

`MESSAGE` ⑤ insertion point. It uses the BGP GeoLoc attribute received over an iBGP session decoded by the first program and sends it to the peer.

To be able to dynamically augment the BGP implementation, the four *x*BGP programs are executed inside a Virtual Machine and are attached to specific *insertion points* in the BGP implementation. An *x*BGP program is composed of eBPF bytecode executed by a user space virtual machine that is included in any *x*BGP compliant implementation. Thanks to this eBPF virtual machine, the same *x*BGP program can be executed on the CPUs used by different router platforms.

An *x*BGP program is not a standalone executable that performs computations autonomously. It can interact with the underlying BGP implementation, access its data structures, and call some of its functions. In contrast with operating system kernels such as Linux, FreeBSD or macOS that expose a similar POSIX interface, there is no standard API for BGP implementations. *x*BGP must then propose a common API to support several BGP implementations. If we take our extension, when the *GeoLoc* program has finished decoding the `Geo_Originator` attribute, it must update the BGP route stored in the BGP implementation. Thus, our extension needs to fetch or set data from the host implementation. For this, the BGP implementation must propose a set of functions, the *x*BGP API [Wira], which enable the interactions between the extension and the internal data structures. For example, with a call to the function `set_attr`, the extension can add a new attribute to the BGP route being processed.

An important data structure of a BGP implementation is its Routing Information Base (RIB). It contains the routes selected by the BGP decision process and pushed in the Forwarding Information Base (FIB). The BGP RIB stores, for each known destination prefix, its BGP route containing its BGP attributes, including its AS-path and the address of the BGP next hop. The RIB also contains information from the intradomain routing protocol such as the cost to reach each next hop. BGP implementations use various data structures to store their RIB. Some implementations simply store the BGP attributes as they were received from the wire like BIRD [CZN20]. Others use a specific structure for each type of attribute as FRRouting [Fou17]. To ensure that the same *x*BGP program can be executed on any compliant implementation, *x*BGP defines its own representation for IP prefixes, next hops, and BGP attributes. For the latter, *x*BGP simply relies on the wire format [RHL06]. *x*BGP also defines a neutral representation of the BGP neighbor's table. With these representations, *x*BGP programs can access the data structures of the underlying BGP implementation. When required, *x*BGP converts the internal representation to its own format before returning data to *x*BGP programs and vice versa.

The remaining of this section describes the composition of the *x*BGP API enabling *x*BGP programs to interact with BGP implementations in Section 4.1.1.

Section 4.1.2 shows how we execute an *x*BGP program inside the BGP implementation. We explain in Section 4.1.3 which challenges we faced to make two different BGP implementations, BIRD and FRRouting, *x*BGP compatible. Finally, we briefly discuss the modifications made to the eBPF Virtual Machine to support *x*BGP in Section 4.1.4.

4.1.1 The *x*BGP API

Besides some utility functions (memory management, conversion between network and host byte orders, simple math functions, etc.), most of the *x*BGP API is specific to BGP [Wira].

To modify internal BGP data structures, *x*BGP programs rely on `getters` and `setters` to access data structures stored on the host implementation. This ensures (*i*) an isolation layer between the host and the *x*BGP program and (*ii*) a uniform method of accessing data regardless of the BGP implementation. These functions convert the internal representation into a universal one understood by *x*BGP programs. In addition, extension codes require access to the BGP internal state (e.g., list of peers, the route attributes, the route next hop). Hence, *x*BGP requires BGP implementations to provide routines translating their internal data structures into *x*BGP ones. These include `getters` and `setters` to access/modify a BGP route including its attributes, next hop and the data that identifies a BGP peer. We also provide functions to iterate the RIB. These enable searching for a route other than those provided by the insertion points, and therefore for searching routes already installed in the BGP routing table.

Existing router OSes do not provide a common way to access internal routing data. The *x*BGP API provides functions to access IGP data, e.g., to retrieve the next hop for routes and use them in use cases described in Section 4.4.

An *x*BGP program can deliberately send a custom BGP message to any peer it wants. Instead of relying on an insertion point to generate the message, the *x*BGP API contains functions to send BGP messages allowing a program to send an urgent message like a BGP notification because the *x*BGP program detected a problem with a given peer.

To access non-standard data such as the geographic coordinates of the router, an extension code may require additional configuration. One approach is to directly include the data inside the code of the *x*BGP program. However, this is not scalable if the operator wants to deploy it on a large number of routers. This induces a recompilation of the code for each of its router. Instead, the *x*BGP API proposes to the network operator to include a configuration data part in a structured textual file accompanying *x*BGP programs called *manifest*. The *x*BGP program uses it later to retrieve what it needs. This extra configuration part is not directly accessible to the *x*BGP program but can be

accessed through a set of API functions.

Finally, *x*BGP programs can be executed as background tasks ⑥ that are called when a timer expires. These tasks are not triggered by a specific BGP event like an insertion point but are rather executed when a timer expires. Background tasks are only used for processes that do not interact with the BGP workflow. Each task controls its timer and *x*BGP deliberately restricts one timer per task to avoid timer explosions. However, the task may ask to queue forever as long as the BGP router is alive. This is particularly interesting for *x*BGP programs that make routine maintenance such as checking for the validity of routing information (e.g. route leaks), refreshing the RPKI cache or aggregating routes to reduce the routing table size, for example. Each background task is executed in a dedicated thread to allow the original BGP implementation to run in parallel. If the *x*BGP program must access or update data, the *x*BGP API must be thread safe. This constraint must be respected when implementing the *x*BGP API.

4.1.2 Executing *x*BGP programs

An *x*BGP program is a set of eBPF bytecodes, either attached to different insertion points or executing background tasks. Each *x*BGP bytecode has its own dedicated memory, including a stack and a heap that are automatically freed after execution. This memory isolation between extension codes is guaranteed by the eBPF virtual machine. This ensures that orthogonal extensions will not interfere with each other. Yet, *x*BGP programs may need to keep persistent storage or to exchange data between the different bytecodes that compose a program. For this, the *x*BGP API provides a key-value store that is similar to the BPF maps used in the Linux kernel.

Each *x*BGP implementation includes userspace eBPF virtual machines that are controlled by a manager. The *Virtual Machine Manager* (VMM) attaches bytecode with an associated virtual machine to one specific insertion point exposed by the host implementation. Each *x*BGP program includes a manifest listing the extension codes and their insertion point. Different extension codes can be attached to the same insertion point, and the manifest defines in which order they are executed. The manifest also lists the different *x*BGP API functions that the bytecode may use.

An *x*BGP program can be attached at different insertion points, i.e., specific code locations in a BGP implementation from where the program can be called. These insertion points correspond to specific operations that are performed during the processing of BGP messages, enabling *x*BGP programs to modify the router's behavior. *x*BGP defines six generic insertion points (green circles in Fig. 4.1) based on the original definition of BGP [RHL06]. The sixth insertion point is dedicated for background tasks.

By default, the VMM only runs one *x*BGP program per insertion point. *x*BGP programs must explicitly tell the host implementation to run the next *x*BGP program (if any) through the `next()` function. This mechanism avoids executing useless code. For example, if we attach two *x*BGP programs that parse different BGP attributes into the insertion point that processes a single BGP attribute, and the first program successfully parses the message, there is no need to run the second one.

4.1.3 Adding *x*BGP to BGP implementations

To demonstrate the feasibility of *x*BGP, we have adapted two open-source implementations: BIRD v2.0.7 [CZN20] and FRRouting v7.3 [Fou17].

Adding the *x*BGP API. Implementing the API induced a total of 400 and 589 additional lines of code [Wirb] on BIRD and FRRouting, respectively. The difference between both is their internal representations of the BGP data structures. The *x*BGP functions that deal with BGP messages and attributes always manipulate them in network byte order (*x*BGP's neutral representation), performing the translation to the storage format used by the implementation if required. FRRouting uses an internal representation that is different from our neutral one. We thus had to implement several functions to do the conversion between the two representations. Another difference is the handling of BGP attributes. BIRD includes a flexible API to manage BGP attributes. *x*BGP simply extends this API. FRRouting does not include such an API, so we had to implement one to be able to manipulate BGP attributes in BGP updates.

Integrating `libxbgp`. `libxbgp` is a portable library, implemented as 432 lines of header code, which consists of two parts: (*i*) the utility functions of the *x*BGP API; and (*ii*) the VMM. The VMM is in charge of executing the right extension code, according to the state of the host implementation. This layer acts as a multiplexer. To include *x*BGP operations, the BGP implementation calls the VMM to execute the associated extension codes. Then, the VMM proceeds as follows. It first checks if there are attached extension bytecodes to the called *x*BGP operation. If not, the VMM executes the default function provided by the implementation. Otherwise, it runs the first extension code mentioned in the manifest. Two outcomes are possible. First, the extension code provides a result for the operation and the VMM returns the output to the caller. Second, the extension code delegates the outcome to another one by calling the special `next()` function. In that case, the VMM checks whether there are remaining codes in the ordered queue. If there are, the VMM runs the next extension code in its virtual machine. Otherwise, the behavior of the *x*BGP operation falls back to the default function provided by the BGP implementation. For instance, two extensions can attach bytecode to the `BGP_RECEIVE_MESSAGE` operation that processes their own dedicated BGP attribute,

calling `next()` once they are done.

Technical challenges. While adding the *x*BGP API and integrating `libxbgp`, we encountered some interesting technical issues. To successfully use the *x*BGP API, data must be available when the function is called. In some cases, data in the host implementation was not available when the insertion point was called to execute the extension code. For example, in FRRouting, export filters are applied to a set of peers sharing the same type of outbound policies. This set is not passed to the code checking the outbound policies but is required to implement the helper function that retrieves data about the BGP peers of the router. We had to write 5 extra lines of code to get the set of peers before calling the insertion point. Also, some data structures were not flexible enough to fully support the *x*BGP API such as the function that adds or modifies a new attribute to a BGP route. However, the internals of FRRouting do not allow adding unsupported attributes that are not defined by any standard (e.g., `ORIGINATOR_ID`). We rewrote this part of FRRouting. To address those issues, we had to add 30 and 10 lines of code to FRRouting and BIRD respectively.

Each API function is called within a context of execution. This context is hidden within the extension code but visible in the host BGP implementation. This makes it possible to control which extension code has called the function. The context is also used to retrieve variables that cannot be directly used inside the extension code. For example, if an extension code needs to allocate extra memory (ephemeral or not), the ephemeral memory is also automatically freed when the extension code terminates its execution. Similarly, the context enables helper functions to access data structures that are out of the extension code's scope. For instance, a dedicated helper function enables an extension to add a new route to the RIB. When setting an insertion point, the BGP implementation can pass a set of arguments. While some are visible inside the extension code, others are not. The RIB function leverages such hidden arguments to access the data structure while being transparent to the extension code.

Limitation of *x*BGP. To better understand what can and cannot be done with *x*BGP, we analyzed the complete list of RFCs, which defines extensions to BGP, that have been published since the publication of RFC4271 [RHL06]. The RFCs can be classified into two different types. (1) The RFCs that modify the original definition of RFC 4271 (6 RFCs) and (2) those that add features on top of BGP (30 RFCs). For (1), *x*BGP cannot be used to implement these types of RFCs because it requires a direct modification of the underlying BGP implementation. For example, increasing the internal buffer size of the BGP message size [BPW19] is not feasible with *x*BGP. For (2), *x*BGP can be used. However, it turns out that our current prototype focuses only on the messages that BGP speakers exchange once the session is established (BGP Updates and BGP Withdraw). Not all session-level extensions to BGP can

be handled by *x*BGP. For example, our current prototype cannot extend the BGP Open or BGP Route-Refresh [Che00] message. However, *x*BGP contains a generic insertion point, `DECODE_BGP_MSG`, that can handle future types of BGP messages. If the underlying BGP implementation does not support route refresh, we can implement it as an *x*BGP program. Modifying *x*BGP to allow it to support the session level could be implemented at a later time, but *x*BGP cannot change the architectural design of the underlying implementation. This is an important limitation of our solution. For example, no *x*BGP program can increase the size of the BGP transmit and receive buffers as defined in the corresponding RFC [BPW19]. The internal structure of an implementation cannot be modified on the fly by a program since the definition of the structures is strongly integrated in the program binaries.

In addition to the limitation of the features that can be implemented, *x*BGP focuses mainly on the internal network, we assume that the network operator enables the necessary *x*BGP programs on the relevant routers. However, if the BGP routers decode an unknown message, it will be silently discarded and will not harm the router but will compromise the other router's computation. BGP capability negotiation messages can be exchanged to indicate whether the extension implemented by the *x*BGP program is supported by both routers implied in the BGP session. Capability support is beyond the scope of this thesis.

Another limitation of the prototype lies in the way background *x*BGP programs are executed. Since the insertion point of the background process runs in its own thread, it is possible for an *x*BGP program to run simultaneously in the main thread and in the thread of *x*BGP background programs which can create race conditions. To prevent this, all insertion points are secured by a mutex, ensuring that only *x*BGP programs linked to a particular insertion point execute before moving on to another. This mutex mechanism effectively manages the execution of *x*BGP programs, preventing the simultaneous execution of multiple *x*BGP programs from different insertion points. Although this method is simple, first it considerably limits interaction between the two threads and second, mutexes imposes a notable overhead on the performances. If the main thread receives a large number of events from the control plane, it has to wait for all background programs to finish, which can cause instabilities in the network. However, background processes are supposed to be executed by the operator infrequently and outside busy periods. This limitation can be resolved in a future version of the prototype by modifying the implementation's internal data structures so that they can be used more efficiently with multiple threads. For example, the BIRD developers have started to revise their implementation to support the execution of multiple threads in parallel [Mat21].

Finally, an operator can inject several *x*BGP programs for the same in-

sertion point. However, this flexibility introduces the possibility of conflicts between these programs. For example, let us consider two *x*BGP programs operating as import filters, where one rejects prefixes more specific than those already present in the RIB, and the other sets a lower local preference for those same prefixes. In this scenario, a conflict arises: one of the plugins wants to accept prefixes, while the other wants to reject them. This creates a non-deterministic behavior that depends on the order in which the plugins are executed. The current prototype does not have the ability to detect conflicts between two *x*BGP programs sending contradictory information. To mitigate the risk of contradictory programs being injected, a possible first step would be to implement advanced program verification methods to monitor the interactions and behaviors of *x*BGP programs in relation to each other.

4.1.4 Augmenting the *x*BGP Virtual Machine

*x*BGP uses the same uBPF virtual machine as the solution presented in chapter 3 to run *x*BGP programs. However, uBPF alone lacked some of the functionality required by *x*BGP. To remedy this, we made changes to the uBPF VM to meet the specific needs of *x*BGP.

- The first change was to enable the execution of the `next()` function. In more recent versions of the Linux kernel's eBPF virtual machine, tail calls can be executed, enabling one *x*BGP program to call another one. However, this functionality was not present in the version of uBPF we relied on, so we implemented it ourselves.

- Second, when examining the `next()` function, we can notice that it did not accept an argument to request the execution of the next eBPF bytecode. Instead, the function is called with a hidden argument containing the execution context. When an API function *x*BGP is called, the VM also passes this hidden argument to it, enabling the API function to identify the bytecode that made the call and take appropriate action. For the `next()` function, the hidden argument is used to call the next correct bytecode to be executed.

- Third, Section 4.2 discusses offline software verification methods to check memory access. However, our modified version of the uBPF VM already includes memory checks at runtime. Therefore, we modify it to enable or disable memory checks at runtime.

- In addition, we introduced controls within the virtual machine to regulate the API functions callable from *x*BGP programs. If the virtual machine detected an unauthorized call, it refuses to load the program

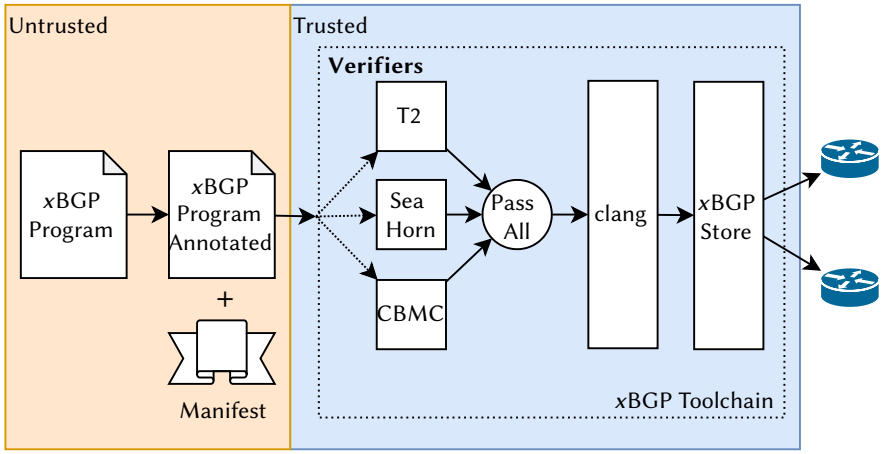


Figure 4.2: High-level view of the *x*BGP verification toolchain.

and the operator is informed via the logging system of the BGP implementation.

- Another enhancement consists of adding support for read-only global variables, which were not initially supported by the virtual machine.
- Finally, we extended support to recent versions of the clang compiler, as the eBPF bytecode produced by these versions was incompatible with the VM version used previously.

4.2 Ensuring the safety of *x*BGP programs

From an operator's viewpoint, injecting an *x*BGP program is always risky since the program will be executed within the BGP implementation. A bug or a malicious code contained in the *x*BGP program can cause a router to crash. A simple approach would consist of letting the VMM monitor their execution and stop them in case of error. This could be too late for errors that could disrupt BGP sessions. Network operators typically need some safety guarantees from the *x*BGP program. The Linux kernel copes with a similar problem by using a custom online verifier [Sta14b] that checks different aspects of eBPF programs *before* they are injected into the kernel.

Verifying *x*BGP programs. *x*BGP also relies on verification techniques to ensure that programs can be safely injected. However, instead of developing a custom verifier [Ger+19], we (*i*) establish a list of properties that an *x*BGP program should respect to be considered as safe and (*ii*) we build a toolchain embedding three existing and well-tested software verification tools allowing the verification of our properties. Our *x*BGP toolchain receives the *x*BGP

programs as input. They consist of C code that uses the *x*BGP API and a manifest provided by the network operator containing the configuration data. This code is by nature untrusted and must be manually augmented with various annotations providing hints to the code verifiers, given the specificities of each one. Such annotated extensions can then enter the *x*BGP toolchain which executes in parallel each verifier. The bytecode is produced only if the code passes all of them. Once the bytecode is generated, it is added to the integrated *x*BGP store. A network operator can safely select and load *x*BGP programs coming from this store. We expect that initially each ISP will have its own store. Later, third parties or router vendors could also develop their own stores. We consider this toolchain as trusted, i.e., we select a particular compiler, `clang`, and specific verifiers, all considered as correct. Therefore, we do not need to reason about the produced bytecode and ignore problems such as handling maliciously formatted bytecode.

Embedded verification tools. The whole *x*BGP toolchain, illustrated in Figure 4.2, is designed to prevent four types of problems that a program can cause. First, if an *x*BGP program enters an infinite loop, it will block the underlying BGP implementation. We use the Terminator 2 (T2) automated termination checker [CPR06] to verify the termination of *x*BGP programs.

The second set of problems concerns the way in which *x*BGP programs interact with the memory of the underlying BGP implementation. As the C language is permissive, it is prone to memory errors. Bugs such as buffer overflow and use-after-release can lead to failures in BGP implementations. Initially, the *x*BGP prototype automatically inserted memory check instructions into the bytecode of the *x*BGP program to check memory bounds. However, this approach had its limitations, as it added extra instructions for the CPU to execute. To overcome this, we opted for an offline approach using CBMC [KT14] and SeaHorn [Gur+15] to check memory-related properties without the need for additional instructions.

The third type of problem is related to *x*BGP and BGP themselves. *x*BGP programs can create new BGP attributes or messages that are sent over a BGP session. *x*BGP programs have two functionalities: sending BGP messages and modifying BGP routes. We verify that the syntax of the generated BGP messages adheres to standards and ensure correct formatting of BGP messages. We use SeaHorn to verify that the BGP messages emitted by *x*BGP programs are fully compliant with the BGP RFCs and that their return values respect the *x*BGP requirements.

Finally, operators may want to be able to impose restrictions on the *x*BGP functions and data structures that a given *x*BGP program can use. For example, a customer filter should only be able to set a `local-pref` value in a chosen range and to change nothing else. So it could never add a new BGP attribute to a route it filters. These restrictions are enforced with (*i*) SeaHorn that checks

Property	Verifier	Type
Termination	T2	Safety
Reads/Writes within *x*BGP program's memory space	CBMC	Safety
No buffer overflow, use after free memory, invalid read, etc.	CBMC	Safety
All strings must be null terminated	SeaHorn	Safety
Correct size/buffer combination	SeaHorn	Safety
RFC-compliant syntax of BGP attributes	SeaHorn	BGP
Valid return value	SeaHorn	Safety
Checking attribute reads/writes	SeaHorn	BGP
Checking API function accesses	`libxbgp`	Safety + BGP
Call the `next()` function to trigger the next *x*BGP program	SeaHorn	Safety

Table 4.1: Properties that *x*BGP programs must satisfy.

the validity of the arguments of the API functions and (*ii*) `libxbgp` which restricts the available API functions at loading time.

To be considered valid, any *x*BGP program must satisfy the properties listed in Table 4.1. If every *x*BGP bytecode satisfies this list, the router is guaranteed (*i*) not to crash and (*ii*) to still follow the definition of the protocol. These properties ensure the local stability of each router. Ensuring the global stability of BGP [GW99; MWA02; GW02a; GW02b] is a problem that goes beyond the scope of this thesis.

Verification macros. Because of their diversity, the verification tools do not offer a common way to annotate programs. In the case of *x*BGP, this would mean annotating the plugin 3 times with different annotations and running the 3 tools manually. For a network operator, manually using several tools can be a long, tedious, and error-prone process. To ease the annotation process, we define a set of multipurpose macros `PROOF_INSTS_*()` abstracting the annotation syntax of the verifiers. Those are only expanded if the corresponding verifier is invoked. When the extension programs are compiled for routers, the annotations are not expanded and thus will not interfere with the normal BGP execution. Figure 4.3 shows an example of such verification macro.

Aside from the verifier syntax abstraction, we mainly bring two contributions. First, we define a set of macros helping network operators to verify the properties listed in Table 4.1. Network operators can use them to annotate their xBGP programs. The macros are translated to their corresponding annotation to the right software verifier. For example, a network operator can use the `BUF_CHECK_*` macros to verify if the BGP attributes sent to a BGP peer are

```
buf[0] = attribute->flags;
buf[1] = attribute->code;
buf[2] = attribute->length;
buf[3] = attribute->data;

CHECK_ORIGIN(buf); // macro
```

(a) Annotated Code.

```
assert(buf[0] == ATTR_TRANSITIVE);
assert(buf[1] == ORIGIN_ATTR_ID);
assert(buf[2] == 1);
assert(((buf)[3] == 0 || \
        (buf)[3] == 1 || \
        (buf)[3] == 2)));
```

(b) Expanded Code (verifier).

Figure 4.3: Example of a verification macro that checks the origin attribute of a BGP route. The macro can be extended or not according to its use. (a) is the original source code and (b) is the code viewed by a verifier.

formatted as stated in the RFCs.

Second, we set up a verification toolchain that automatically performs verification on the xBGP programs. It automatically and transparently calls all the verification tools and verifies the annotations contained in the source code of the programs. If all properties are satisfied, the system stores the verified plugins in a "plugin store", which the programmer or network operator can use to inject into their routers. The routers will only accept plugins that have been verified and signed by the plugin store.

Those macros, in conjunction with our verification toolchain, allow a complete abstraction of the verification process. This makes the usage of *x*BGP simpler for network operators. The entire set of verification macros is defined in the corresponding publication [Wir+23a].

The remainder of this section focuses in more detail on two aspects of the verification that the xBGP toolchain uses: T2 and the dynamic verification performed by libxbgp at runtime. For more details on Seahorn and CBMC, please refer to the corresponding publication [Wir+23a].

4.2.1 Proving *x*BGP Programs' Termination

T2 (TERMINATOR 2) is a program analysis tool for termination [CPR06] and temporal property [Bro+16] verification. We were successfully able to prove the termination of every *x*BGP program that implements the use cases defined in this chapter. Table 4.3 reports the total time taken by the verification toolchain to verify all the properties defined for the *x*BGP programs, including the termination checks. However, to check the termination we had to slightly modify the source code since some specific features of the C language were not supported by the prover. First, when using fixed-width integer types (e.g., `uint8_t`), T2 was not able to generate the proof of termination. We had to convert those types to their primitive type. Second, all the loops of the program must be explicitly bounded. For example, if the *x*BGP program needs to parse a BGP attribute, we must explicitly bound it to 4096 iterations, the

maximum size of a BGP message [RHL06]. Third, T2 does not handle bit shift operations. To solve this issue, we encapsulated the bit shift computation in a non-deterministic function. This is a function that is not defined in the source code of the *x*BGP program but simply tells T2 that it returns an arbitrary integer value that T2 can handle. Such non-deterministic function is also considered to terminate by T2.

4.2.2 Enforcing Operator-Imposed Restrictions

Thanks to the manifest, the operator can list the *x*BGP API functions and the data structures that each *x*BGP program can use. Imagine a filter that only checks the validity of the route without modifying any data related to this route. To decrease the risk of introducing bugs in *x*BGP programs, the operator can restrict the set of API functions the program can call. In this example, the filter should have a read-only view, and thus should not call any function altering BGP data structures.

To settle this, we implemented a permission manager inside `libxbgp` that verifies, at load time, the functions that a given *x*BGP program calls according to its manifest. Just before being loaded, `libxbgp` checks the *x*BGP bytecode to look for unauthorized API function calls.

Network operators use BGP communities [DB08; Str+18] to enable their customers to activate specific features such as setting `local-pref`, AS-path prepending, or selective advertisements on a per-route basis. With *x*BGP they could provide even more advanced services. Imagine you are a network provider that proposes to attach filters developed by its clients to their eBGP sessions. You define a set of BGP attributes the client can modify such as MED, `local-pref`, etc. When they use communities, operators establish policies on the attributes which can be modified in a BGP route. For example, they define ranges of possible values for the `local-pref` attribute [DB08]. To modify an attribute for a route, an *x*BGP program calls the `set_attr` API function. When the *x*BGP toolchain processes such a program, SeaHorn verifies if the arguments of the API functions respect the policies defined in the manifest, i.e., if both the argument to change and its new value are legitimate. This is done by adding assertions in the source code of the *x*BGP program supplied by the customer.

4.3 Overhead of the current *x*BGP prototype

Using *x*BGP in BGP implementations brings flexibility for network operators since they can use a simple abstraction to program their router. However, this flexibility has a price in terms of performance. To evaluate the overhead of `libxbgp`, we consider three different features that are already implemented

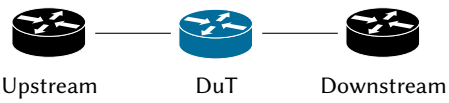


Figure 4.4: Simple network used for *x*BGP evaluations.

in both native FRRouting and BIRD to have a fair comparison with *x*BGP. The first is a simple filter adding an arbitrary MED value to all exported routes. The second provides support for extended communities [TSR06]. The third is a complete implementation of Route Reflection [CBC06]. While we expect operators to mostly develop simple plugins such as the first two, the Route Reflection extension demonstrates the of flexibility *x*BGP by covering the whole BGP workflow described in Section 4.1. Furthermore, since Route Reflection is supported by both FRRouting and BIRD, this extension enables us to compare the overhead of an *x*BGP implementation with native ones.

To evaluate the performance impact of *x*BGP, we use the simple network described in Figure 4.4. We measure the delay between the first BGP update sent by the Upstream router and the last update received by the Downstream one. This reflects the time needed for the Device under Test (DuT) router to process the routes sent by the Upstream router. The Upstream and Downstream routers are running an unmodified implementation of BIRD v2.0.8 while the DuT router is running the *x*BGP version of BIRD or FRRouting according to the test. The DuT router is running an Intel® Xeon® X3440 @2.53GHz with 16 GB of RAM, Linux kernel v5.15.29 and Debian 11.

The Upstream router sends a full routing table from a recent RIPE RIS snapshot (`rrc0`, June 3, 2021, at 4:15 PM) containing 873k IPv4 routes and 120k IPv6 routes. We consider multiple executions of the BGP daemon located in the DuT router. Table 4.2 shows the relative performance impact of running the extensions with *x*BGP programs compared to their native implementation in both BIRD and FRRouting. For each *x*BGP compatible implementation, we run 10 times the *x*BGP programs and compute the convergence time. The convergence time is the time between the first BGP update message received from Upstream to DuT and the last BGP update message sent from router DuT to Downstream.

Before even loading any *x*BGP extension, bringing support *x*BGP in a BGP implementation involves an initial overhead. More specifically, the host implementation must first construct the argument to be passed to the *x*BGP program, then request the execution of the corresponding insertion point, and finally execute the *x*BGP termination routine. These additional steps increase the total number of instructions to be executed compared to the native non-*x*BGP implementation. To quantify the cost of adding `libxbgp` takes to BIRD and FRRouting, we ran both implementations of *x*BIRD and *x*FRR without

Use Case	Processing Time	
	xFRR	xBIRD
No *x*BGP program	+1.05%	+1.6%
Filter Set MED	+6.67%	+2.59%
Extended Communities	+5.93%	-0.67%
Route Reflection	+12.97%	+7.43%

Table 4.2: Performance impact of running *x*BGP programs to *x*BIRD and *x*FRR.

plugins and compared them to their non-*x*BGP compatible versions. Making both implementations of *x*BGP compatible adds a cost in the convergence time of 1% and 1.6% in FRR and BIRD respectively.

We now consider the MED filter (one insertion point) and the extended communities (two insertion points) extensions. When implemented as *x*BGP programs, these slightly increase the convergence time compared to their native version. The Just-In-Time compiler used inside the virtual machine does not optimize as efficiently as the one producing x86_64 native code. In particular, computation-intensive bytecode involving additions, subtractions, and multiplications take 50% more time to run than native code. This overhead is even worse when considering division and modulo operations.

Yet, we observe a higher convergence time increase for FRRouting than BIRD. By analyzing the execution of each *x*BGP bytecode with a code profiler, we identified two main reasons for this difference. First, to communicate with the host implementation, the *x*BGP program must pass through a dedicated *x*BGP API. For security reasons and because of the internals of `libxbgp`, the data of the host implementation are first translated into a neutral representation, then copied into a dedicated memory area, accessible in writing and reading by the bytecode. Translation and copying play an important role in the execution of a plugin but are needed to run the same *x*BGP program in several BGP implementations. BIRD internally uses data structures that are closer to the *x*BGP neutral representation than the FRRouting ones, hence involving less translation overhead. Second, FRRouting and BIRD have different internal architectures. The interactions between the `libxbgp` API and the BGP implementations are different. FRRouting is less flexible as its implementation is not designed to be quickly extended with new functionalities. While in BIRD, most of the insertion points map to a specific place, in FRRouting some insertions points must be repeated at different code places, involving up to four times more *x*BGP program calls than using BIRD.

We now consider the Route Reflection extension covering the whole BGP workflow. Supporting this feature requires a list of all iBGP client peers.

Use Case	C LoC	eBPF Insts	Total Verif Time(s)
Geo TLV (§4.1)	388	1340	664
MED Filter (§4.3)	55	149	79
Extended Communities (§4.3)	196	322	86
Route Reflection (§4.3)	509	3853	27
Route Selection (§4.4.1)	62	148	27
Zombie Detection (§4.4.2)	1071	5697	277
Decision Monitor (§4.4.3)	306	437	29
Propagation Time (§4.4.4)	560	805	73
Valley Free (§4.4.5)	143	960	182
Prefix Origin (§4.4.6)	150	661	57
IGP Data (§4.4.7)	36	149	3

Table 4.3: Verification of the *x*BGP programs supporting our use cases.

Routers' implementations use their dedicated CLI syntax to define all their iBGP client peers. `libxbgp` does not have access to this CLI configuration since it is implementation-dependent. Instead, it relies on its configuration data within the manifest that can be accessed at any time by the *x*BGP program. The topology used is the same as that described in Figure 4.4. The Upstream and Downstream routers are Route-Reflector (RR) clients of the DuT router, which acts as the RR. On average, BIRD's convergence time is 7.5% slower than the native code while FRRouting's one is 13% slower. The previous elements still hold to explain the difference between BIRD and FRRouting. In particular, there are more calls to *x*BGP programs in FRRouting due to its code architecture than in BIRD (`BGP_ENCODE_MESSAGE` is called 4 times more), and the translation time to convert data structures is non-negligible in FRRouting (up to 40% overhead for the import filter). Still, the performance overhead of *x*BGP remains within acceptable bounds.

4.4 Use Cases

Section 4.1 presented the *GeoTLV* feature to demonstrate that *x*BGP programs can create new attributes that influence the router. Section 4.3 presented the MED filter, extended communities, and route reflection to make a performance comparison. This section presents other use cases, which are not implemented natively in FRRouting and BIRD. They illustrate the advantages of *x*BGP for various classes of problems that operators want to solve. It is true that the features of this section could be implemented in any BGP implementation without *x*BGP. However, feature support depends on the pace of implementation by all vendors. Thanks to the *x*BGP design, an operator can quickly design

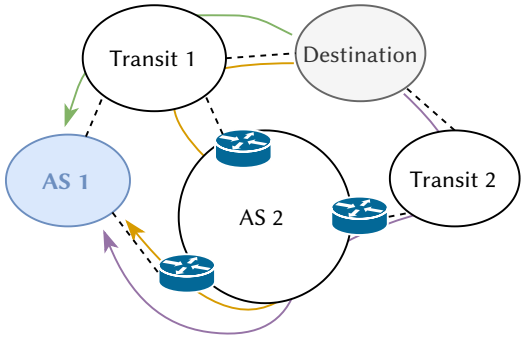


Figure 4.5: Path Diversity in a Network.

its features and introduce them into the network before they are implemented by the vendor. xBGP is the first step to bring extensibility to the network. The first use case defines an *x*BGP program (Section 4.4.1) to influence the decision process and the import and export filters from the BGP client point of view. The second use case detects zombie routes (Section 4.4.2). These are routes that are installed in the routing table but are no longer reachable. Third, operators always try to understand the state of their network to improve it as much as possible. We present two use cases (Section 4.4.3 and 4.4.4), where BGP is monitored using communities. The fifth use case is related to route filtering in data-centers (Section 4.4.5). It demonstrates that *x*BGP can provide a programmable interface to design complex import and export filters. Our sixth *x*BGP program (Section 4.4.6) gives another example of a special filter that checks the origin of a route. Finally, our seventh use case (Section 4.4.7) shows that an *x*BGP compatible implementation can leverage IGP information to make routing decisions.

Table 4.3 reports the size of the *x*BGP bytecode, the number of lines of code and the time taken to validate every *x*BGP program according to the properties defined in Section 4.2.

4.4.1 Customer Selecting Routes

A BGP router only selects one route for each prefix even though it learns multiple routes. As a result, it will only send one route to each BGP neighbor, which decreases the path diversity. Consider Figure 4.5 to illustrate the situation. AS1, a multihomed stub network having peering links with Transit 1 and AS2. We are interested in the propagation of the routes to the destination network depicted in gray. To maximize path diversity in AS1, it should learn the purple path from AS2 to leverage the two different transits. However, AS1 cannot influence the decision process of AS2's routers.

Enabling the dissemination of multiple routes can bring several benefits

such as load-balancing [LT21], avoiding route oscillation [GW02a] and faster local recovery upon a network failure [SB10]. With *x*BGP it becomes possible to influence the border router to announce the route the client prefers. To design such an *x*BGP program, all edge routers must enclose their BGP client to one Virtual Routing and Forwarding table (VRF) [Van09]. All the routes learned from all neighbors will be exported to the main BGP-VPN RIB's router and then exported to the VRF of each client so that they can have a full view of the routes. Since all clients are in their respective VRFs, the BGP decision process is different for each of them and can therefore be influenced by an *x*BGP program that decides which route to advertise.

We designed a simple *x*BGP program that randomly selects one of the available routes in the VPN RIB thanks to the *x*BGP API function `get_vrf`. It demonstrates that *x*BGP allows the operator to create a customized and more powerful route selection compared to the traditional router CLI. Accessing the VPN RIB through a simple router configuration is something that cannot be done with traditional BGP implementations. Furthermore, an *x*BGP program has access to the entire BGP route and the internal data structure of the BGP router. *x*BGP therefore provides greater flexibility compared to the classical CLI.

We were successfully able to check the termination with T2, the C errors with CBMC, its compliance to the *x*BGP return values. We also verified that the program does not use API functions altering the BGP internal state.

4.4.2 Detecting BGP Zombies

When a route becomes unavailable, a BGP router sends a withdraw message to all its peers. Because of software bugs [Fon+19], it may happen that one of these BGP peers fails to process such a withdraw message. As a result, the route is still considered reachable by the failed router. This is an operational problem because the withdrawal is not propagated, and part of the network still believes that the route is available. If packets still follow this zombie route, they will be blackholed. Measurements indicate that these zombie routes are common and affect many ASes [Ong+21].

To detect persistent zombie routes, we designed an *x*BGP program that is executed periodically. It uses the timestamp of the arrival of a route in the RIB to detect the routes that are older than x days. Our threshold is arbitrarily fixed to a day. Our *x*BGP program is configured to be executed during the maintenance window. It parses the entire BGP RIB thanks to the API functions `*_rib_iterator`. If a route is older than the configured threshold, it is flagged as a possible zombie. To confirm the status of the route, the router needs to request it again from the peer that announced it. This could be done with standardized mechanisms such as Graceful Restart [Rek+07]

or Route Refresh [Che00; PCV14]. However, those two approaches require the remote router to announce again its entire BGP routing table. For the sake of performance, we decided to only ask the remote peer to reannounce the routes flagged by the *x*BGP program. We introduce a new type of BGP message called `BGP Refresh`. It contains a list of prefixes that the router wants to confirm. The peer receiving the `BGP Refresh` message will announce a withdraw or an update message if the routes are not available anymore or still in its BGP routing table respectively. *x*BGP allows sending BGP messages via the `schedule_bgp_message` API function.

It is difficult with a traditional BGP implementation to detect such a zombie route. Indeed, there is no mechanism to analyze and perform an action according to the state of the BGP routing table. To include this feature, the network operator must convince each router vendor to add this feature into its implementation. This use case demonstrates that *x*BGP can outperform the current configuration method that is proposed in classical BGP implementation.

This *x*BGP program successfully passes the T2 and CBMC verifications. As it manages BGP messages, we verified their compliance to the RFC. We also checked that the size of the buffers announced to the API function matches their real size. This *x*BGP program is an example of functionality that cannot be performed using the traditional router CLI while the router is running.

4.4.3 Monitoring the BGP Routing Decision

Currently, if a network operator would like to debug its BGP routers, they only have monitoring information from the routers they directly control. This is due to the fact that the traditional BGP specification only provides the exchange of local routing information but does not provide any abstraction to send monitoring information about the routing process. Yet, a support of a dedicated monitoring channel has been proposed [Sha+11] but this is still not implemented on all vendor's routers. More recently, the BGP Monitoring Protocol (BMP) [SFS16] enables a BGP router to establish a connection with another BGP router to collect a wide range of metrics. However, two notable limitations need to be taken into account. First, BMP does not offer sufficient flexibility to monitor fine-grained parameters, potentially limiting the information that can be monitored. Second, operators may be reluctant to export a monitoring interface to another operator because of concerns about exposing sensitive information. Indeed, they cannot fully control what BMP transmits to another AS router. In our solution, a BGP router can ask its neighbor to give different metrics such as its number of reachable prefixes, its Adj-Rib-In, its current state, etc. By implementing an *x*BGP program to collect specific metrics, the network operator is able to provide only the information requested by another operator, thus avoiding the exposure of unrelated data.

In addition, the use of verification tools can serve to reinforce this concept by ensuring that data does not inadvertently leak extraneous information.

Some router vendors actually provide commands to retrieve the local state of a router. However, information is restricted to the router view only and does not include the status of routers that are outside the operator's management scope. Having statistics from other routers could bring new possibilities. For example, if an operator encounter routing problems or suboptimal routing, they can use these statistics to pinpoint which specific BGP path selection criteria are causing the issues. For example, if the length of the AS-Path is causing problems, they may need to check their BGP connections and routing policies. This can be done by adjusting the local preference value or using AS-Path prepending to achieve their routing goals.

We leverage *x*BGP to instrument the BGP implementation to retrieve at which step of the BGP decision process the route has been chosen. Each time it runs, an *x*BGP program retrieves the reason of the decision in the process. It can be retrieved through the arguments passed to the *x*BGP program. To inform other BGP routers, this information is added as a BGP community when sent to other BGP speakers. This is done by the API function `set_attr_to_route`. This way, other routers can parse and use this information to adapt their routing strategies. This *x*BGP program also collects statistics about the other steps of the BGP decision. Each time a route is selected, the *x*BGP program increments an internal counter. It repeats the operation for each decision step. When a route is sent to any peer, these statistics are attached as a community. The BGP router receiving the statistics can have a broader view of the current routing table of its peer. As this information cannot be obtained via the router's conventional command line interface (CLI), this new approach can facilitate more accurate routing decisions. It enables network operators to efficiently monitor, troubleshoot, optimize and plan their BGP routing with greater granularity. Indeed, this new form of active monitoring extends the capabilities of traditional monitoring tools such as BMP, SNMP, etc., as these tools do not modify the BGP messages they transmit to BGP neighbors. As a result, network operators now have access to valuable external information that was previously inaccessible to them, enhancing their network management capabilities.

This *x*BGP program successfully passes all the verifiers. Since it handles the BGP community attribute, we also verified that the format is respected according to the corresponding RFC.

4.4.4 Measuring BGP Route Propagation Times

For mission critical systems, the convergence time of a routing protocol is an important metric. It helps to better understand what could be the cause

of a slow convergence. Discussions with network operators indicate that commercial router vendors provide undocumented CLI commands to access profiling points. However, this profiling information is local to each router. It could be useful to exchange such information within an entire network. This could open new opportunities to better understand the current state of the network. One example of such monitoring is the time taken by a BGP route to traverse an AS. To support such monitoring information, BGP must be augmented to add in each route its arrival time at each AS border router. Our *x*BGP program defines a new non-transitive BGP attribute, called `RECEIVED_TIME`. It adds this attribute when a route is received over an eBGP session (thanks to the `set_attr` *x*BGP API function family). It traverses the AS with the BGP route until it reaches an edge router. The `RECEIVED_TIME` attribute is removed when the associated route is sent over an eBGP session and the border router computes the difference between its current NTP time and the one of the attribute. As for the previous use case, exchanging such monitoring information is not currently feasible with traditional routers. These two use cases show that *x*BGP can perform a new type of active monitoring by exposing the internal data of the BGP implementation itself to inform the other neighbor of the current BGP routing state.

4.4.5 BGP in data centers

Although BGP was designed as an interdomain routing protocol, it is now widely used as an intradomain routing protocol in data centers [LPM16]. This is mainly because BGP scales better since it does not rely on flooding in contrast with OSPF or IS-IS. Another benefit of BGP is its ability to support a wide range of configuration knobs and policies. However, BGP suffers from several problems that force the network operators to tweak their BGP configurations [LPM16]. These tweaks make BGP configurations complex and more difficult to analyze and validate [Bec+17b]. To illustrate this complexity, let us consider the data center shown in Fig. 4.6. Routers S1 and S2 are the Spine routers, L10 ... L13 the leaf routers, and T20 ... T23 the top-of-the rack routers. In such a data center, there is no direct connection between the routers at the same level in the hierarchy. Data center operators usually want to avoid paths that include a valley (e.g., L10 → S1 → L11 → S2). To achieve this, they usually run eBGP between routers, but configure the same AS number on S1 and S2 (even if these routers are not directly connected). Similarly, L10 and L11 (resp. L12 and L13) use the same AS number. With this configuration, when S2 receives a BGP update with an AS-Path through S1, it recognizes its AS number and rejects the route. This automatically blocks paths that include a valley and also helps to prevent path hunting.

Unfortunately, using the same AS number on separate routers can cause

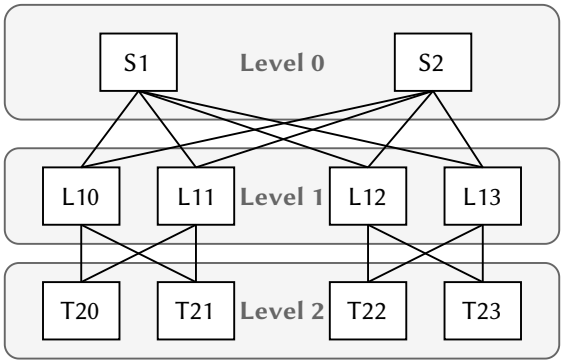


Figure 4.6: A simple data center.

problems. First, operators can no longer look at the AS Paths to troubleshoot routing problems since different routers use the same AS number. Second, by prohibiting valley-free paths, the operator implicitly agrees to partition the network when multiple failures occur. Consider again Figure 4.6. If both links L10-S1 and L13-S2 fail, then the only possible path between L10 and L13 is L10 → S2 → L12 → S1 → L13. If the same AS number is used on S1 and S2, this path will never be advertised.

With *x*BGP, a network operator can use different AS numbers for their routers and implement specialized filters on the spine and leaf routers. For example, if S1 and S2 are both connected to transit providers and can reach the same prefixes, then L10 should never reach S2 via S1 and L11. However, this path should remain valid if the final destination is a prefix attached below L13.

To implement such a filter, we load a manifest containing every eBGP session from a router of level i to a router of level $i + 1$ in a pair having the following form: $(AS_{li}, AS_{l(i+1)})$. For each route, the filter checks each consecutive pair of the AS-Path. If a pair of this manifest is included in the AS-Path, the filter rejects the route since it is not valley-free.

This *x*BGP program successfully passes T2 and CBMC checks. SeaHorn confirms its *x*BGP compliance relative to its use of the API functions and the return values.

4.4.6 Validating BGP Prefix Origins

The interdomain routing system is regularly affected by disruptions caused by invalid BGP advertisements originated from ISPs. Examples include the AS7007 incident in 1997, the announcement of a more specific prefix covering the YouTube DNS servers by Pakistan Telecom in 2008, or BGP prefixes leaked by Google in 2017 that disrupted connectivity in parts of Asia. These problems

and many similar ones were caused by configuration errors.

To cope with these (mainly manual) errors, network operators and the IETF developed three types of solutions. First, they enhanced the address registries to include cryptographically signed certificates that associate IP prefixes to origin ASes. This is the basis for the Resource Public Key Infrastructure (RPKI). Thanks to the RPKI, an operator can verify whether ASx is a valid originator for prefix $p1$. Second, the SIDR working group developed techniques to allow a router to query the RPKI to validate the origin of the routes that it receives [HM12]. The work on validating the origin of prefixes started in 1999 [Lyn99], the first RFC was adopted in 2012. It is slowly being deployed [Reu+18; Chu+19]. This approach is now being extended to also use the RPKI to validate other elements of the AS-Path [Azi+20]. However, this approach has not yet been implemented in most BGP routers. The third long-term solution, which should also cope with malicious BGP hijacks, will be to extend BGP to carry digital signatures inside the BGP messages [LS17]. This extension is far from being deployed.

To illustrate the flexibility of *x*BGP, we consider a RPKI-based route origin validation variant. The network operator includes in the configuration data of the manifest all prefixes it knows the origin. We assume the operator has themself validated the ROA (Route Origin Authorizations) signatures before generating the file. This file is used by the *x*BGP program each time a BGP route is received by a peer to check if the origin AS of the route matches the one contained in the file.

To evaluate the performance of our prefix origin validation, we use the same testbed as in Section 4.3 except that we use eBGP sessions for links L1 and L2. Our *DuT* does not implement the RPKI-Rtr protocol [BA13; Wäh+13] but loads configuration data that considers 75% of the injected prefixes as valid. For this test, our extension code checks the validity of the origin of each prefix but does not discard the invalid ones.

We compare our extension codes running on BIRD and FRRouting to their native implementations without any prefix validation. We do not compare our solution with the RPKI-Rtr protocol since we do not totally implement the RPKI protocol. We only check the origin of the route. It takes up to 12% and 14% more time to converge a complete routing table by running the prefix validation plugin for xFRR and xBIRD respectively. The difference in execution between the two implementations is also explained by the difference in the internal representation of the data structures used.

The termination and absence of C errors were proved with T2 and CBMC. SeaHorn also confirms that the *x*BGP program does not write any data in the memory of the host implementation and its compliance on the return values.

4.4.7 Filtering Routes Based on IGP Costs

Since the *x*BGP API provides access to the data structures maintained by a BGP implementation, network operators can leverage it to implement new filters. As a simple example, consider an ISP having a worldwide presence that wants to announce to its peers the routes that it learned in the same continent as the advertising BGP. This policy can be implemented by tagging routes with BGP communities on all ingress routers and then filtering them on export. While being frequently used [DB08], this solution is imperfect. Consider an ISP having two transatlantic links terminated in London, UK, and Amsterdam in The Netherlands. This ISP has a strong presence in Europe and two links connect the UK to other European countries. If these two links fail, packets between Germany and London will need to go through Amsterdam, the USA, and then back to the UK. When such a failure occurs, the ISP does not want to advertise the routes learned in the UK to its European peers. With BGP communities, it would continue to advertise these routes after the failure.

Using the *x*BGP API, the operator could implement this policy as follows. First, they configures the IGP cost of the transatlantic links at a high value, say 1000 to discourage their utilization. Second, they implements a simple export filter that checks the IGP cost of the next-hop before announcing a route. The complete source code of such a filter is shown in Listing 4.1. It is attached to the `BGP_OUTBOUND_FILTER` ④ insertion point. If the IGP cost to the BGP next hop distance is acceptable, the function calls the special function `next()`. This informs the VMM to execute the next bytecode attached to the insertion point. If the extension code is the last to be executed, the insertion point proposes to fall back to the native code. To reject the route, the extension code returns the special value `FILTER_REJECT` to the host implementation.

For this *x*BGP program, we used SeaHorn to ensure return values were meaningful to `libxbgp`. T2 and CBMC are also used to check the termination and the absence of any C errors. We also verify that the *x*BGP program has only a read-access to the host implementation.

4.5 Related Work

Protocol programmability. In the late nineties active networks were proposed as a solution to bring innovation back inside the network that was perceived as being ossified [TW07]. Most of the work in this area focused on the possibility of placing bytecode inside network layer packets. PLAN [TW02], ANTS [WGT98] and router plugins [Dec+98] are examples. In the control plane, researchers built upon this idea to propose new solutions such as the 4D architecture [Gre+05], the Routing Control Platform that centralizes routing [Cae+05] or Metarouting [GS05] that proposed to open the definition of

```
uint64_t export_igp(bpf_full_args_t *args UNUSED) {
  struct ubpf_nexthop *nexthop = get_nexthop(NULL);
  struct ubpf_peer_info *peer = get_peer_info();
  if (peer->peer_type != EBGP_SESSION) {
    next(); // Do not filter on iBGP sessions
  } if (nexthop->igp_metric <= MAX_METRIC) {
    next(); // the route is accepted by this filter;
  }         // next filter will decide to export route
  return FILTER_REJECT;
}
```

Listing 4.1: An example of export filter rejecting BGP routes having a too large IGP nexthop metric.

routing protocols using a declarative language. While these previous works propose configuration languages or centralized approaches to deal with network programmability, *x*BGP relies on an existing decentralized control plane protocol on which an operator can add its new functionality to locally influence the routing.

Bringing flexibility to an implementation of a network protocol has been studied in the literature. Researchers have proposed using extension codes to extend transport protocols like STP [Pat+03], QUIC [De +19] and the FRRouting implementation of OSPF and BGP in Chapter 3.2. However, the architecture of these pluginized approaches is close to the internal architecture of a single protocol implementation and does not offer the flexibility to pluginize different implementations of the same protocol. *x*BGP goes one important step further by enabling very different implementations to execute the same *x*BGP program. *x*BGP tries to determine what all implementations of a protocol have in common to try to find a common usable interface.

To ease the automation and the configuration of their devices, routers vendors added scripting languages that enable the network operator to execute recurrent tasks [BDL10]. However, this acts as a simple shell that cannot be used to extend the router implementation. Other vendors integrated the python language into their router OS [Jun21] to perform automation task more easily, such as configuring the router or executing a monitoring routine when a particular event occurs.

Reducing the BGP implementation to its minimum has been studied with CoreBGP [Whi20]. However, it only manages the basic BGP Finite State Machine on which plugins written in the Go language are inserted. The remainder of the BGP logic such as sending BGP messages or managing the routing table is passed to the plugins. CoreBGP plugins react to an FSM events while *x*BGP programs react to protocol events defined by the insertion points depicted in Figure 4.1.

XORP [HHK03; Han+05] was introduced to propose an open-source software router platform. This solution has been designed to allow researchers to easily develop their own extensions to a routing protocol. Other open-source routing stacks have been developed such as Quagga [Ish+], FRRouting [Fou17] or BIRD [CZN20]. While these open-source stacks allow modifying the source code of the routing software, *x*BGP goes one step further by introducing a simple API to interact with the routing software. There is no need to look directly in the code of the implementation to understand how to integrate an extension. Anyone who wants to add their own extension will interact with the router through *x*BGP. Throughout this chapter, we demonstrated that an *x*BGP extension code written only once can be successfully executed by two open-source routing stacks, FRRouting and BIRD.

Virtual Machines. `libxbgp` is based on a user-space implementation of the kernel eBPF VM [IO 18]. In recent years, Linux kernel developers have integrated a virtual machine called eBPF [Sta15] which enables programs to inject executable bytecode at specific locations inside the kernel. It was initially targeted at monitoring kernel operations [Gre+05], but also for fast packet processing [Gre+05]. Researchers have used eBPF to support networking programming with IPv6 Segment Routing [XDB18] and extend TCP [TB19]. Other frameworks could have been used such as WebAssembly [Haa+17] or Lua [Ier16] that is widely used in industrial systems. Using another type of VM can be studied to measure its performance and its relevance to routing protocols.

Verification tools. The PDS (Plugin Distribution System) [Ryb+21] provides secure verification and distribution of extension code for Pluginized QUIC [De +19]. It allows the automation of different types of verification for several extension codes at the same time. Our *x*BGP toolchain includes more verifiers and checks more properties. While the PDS uses a Merkel tree to secure the distribution of plugins, the *x*BGP toolchain simply keeps them in a store that is used by the network operator.

4.6 Conclusion

We presented *x*BGP, a new paradigm that enables network operators to innovate in routing protocols. *x*BGP allows them to write their extensions or modifications in the form of an *x*BGP program that can be executed inside the protocol implementation. This programmability could help network operators innovate with existing distributed routing protocols as Software Defined Networking lead to the development of programmable switches. Our solution has been proposed for BGP but could also be adapted to support other routing protocols. We further introduced the *x*BGP toolchain that allows operators to annotate *x*BGP programs to verify their safety. It checks if the *x*BGP pro-

gram meets the local properties of the router such as the termination, the memory constraints and if the *x*BGP program meets the definition of BGP. If it passes the verification step, the *x*BGP program can be safely added to the BGP implementation and is guaranteed not to corrupt the router. Finally, we demonstrated *x*BGP's capabilities by proposing several use cases that have been implemented with our solution. Among them, *x*BGP enables the operator to add new attributes to a BGP route, implementing complex filters, allowing a client to influence the BGP decision process and executing background tasks.

Part III

Revisiting the Transport Layer Used by Routing Protocols

The Benefits of Secure Transport for Routing Protocols

5

At the time of writing, the work presented in this chapter is unpublished yet.

Part II of this thesis focused on improving routing protocols by using a virtual machine to introduce extensions in the protocol. This is one approach to enable operators to achieve protocol extensibility.

In routing protocols, the transport of routing messages is an important aspect that requires improvements and innovation. This part of the thesis therefore aims to enhance the transport of routing messages within routing protocols. These messages play a crucial role in sharing information about best routes to specific IP prefixes. To guarantee the reliability of this information exchange, a reliable transport protocol is essential. However, since the standardization of routing protocols in the late 1980s, there has been little innovation in this domain. During that time, security was not a major concern, as the Internet was mainly used for research purposes and had limited commercial impact. Moreover, the limited hardware resources of routers at that time restricted the adoption of other reliable transport protocols and the implementation of secure routing sessions.

Intra-domain routing protocols such as IS-IS and OSPF have their own customized transport protocols for reliable delivery over point-to-point links or local area networks (LAN). However, their current reliable transport mechanisms are not very efficient. They rely heavily on timers to avoid overwhelming neighboring routers. While this limited form of flow control may have been effective in the past, modern routers are more efficient and capable of handling higher volumes of routing messages. As a result these timers and this basic flow control slow down the exchange of link state packets. The IETF is currently investigating mechanisms to speed up the flooding [Dec+23] of IS-IS link-state packets. Also, informal discussions took place on the IETF mailing lists to propose replacing the IS-IS transport layer with TCP [SV18]. However, such change raised several problems. First, routers have to use a TCP/IP stack to exchange routing information. Second, there were concerns

about head-of-line blocking. Pending prefixes in the TCP stream will not be processed until the last lost message has been retransmitted, which slows down network convergence.

When the Border Gateway Protocol (BGP) was designed, TCP was chosen as the transport protocol because of its well-established nature. Since BGP uses TCP, it automatically benefits from decades of TCP optimizations. Although it is possible to replace TCP with an equivalent protocol, a full evaluation of such a replacement has not yet been carried out to the best of our knowledge. Nevertheless, BGP over TCP also suffers from certain limitations. It lacks the security features needed to protect BGP sessions from various attacks [Mur06].

In this chapter, we propose to replace BGP transport with QUIC [IT21], a secure UDP-based stream protocol that features both reliable transport mechanisms and end-to-end data encryption. While the original work mentioned in this chapter also covers transport layer replacement in IGPs, this chapter focuses exclusively on BGP. More detailed information on OSPF transport layer replacement can be found in the corresponding work [Wir+23b].

The chapter is then organized as follows:

- We start explaining QUIC in Section 5.1 and by motivating in Section 5.2 the need to replace the transport protocols used by the routing protocols.
- In Section 5.3, we discuss the potential advantages of using QUIC as the transport layer for routing protocols, such as improved security and the introduction of new transport features.
- Section 5.4 provides insights on the of BGP over QUIC prototype. We also discuss performance considerations and the evaluation setup used for the rest of the chapter. The next section presents several use cases enabled by QUIC that we evaluated.
- In Section 5.5, we show how QUIC can improve the flexibility of BGP:
  - First, we detail how embedding additional routing information in X.509 certificates can facilitate secure session configuration.
  - Second, we discuss how QUIC enables on demand BGP session establishments.
  - Third, we show that leveraging QUIC features allow improved blackholing services.
- We then overview in Section 5.6 the related works.
- Finally, we conclude this chapter in Section 5.7.

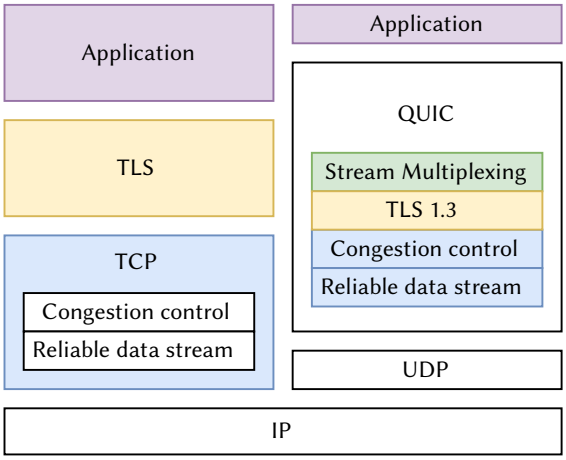


Figure 5.1: Comparison of TCP+TLS and QUIC network stacks.

5.1 The QUIC Transport Protocol

Before explaining why QUIC can be used to transport routing protocols, we briefly introduce this protocol in this Section.

Originally developed by Google [Ros13], then standardized by the IETF in May 2021 [IT21], QUIC has gained in popularity and is increasingly used on the Internet. Major service providers such as Cloudflare, Google and OVH offer various services using QUIC [Zir+21].

Figure 5.1 describes the QUIC protocol in the network stack, with a comparison with a TCP+TLS stack. Like TCP, QUIC is designed for reliable data exchange, with built-in congestion control mechanisms. However, QUIC goes further than TCP by integrating functionality for several layers of the network stack. It operates on top of UDP [Pos80] and directly incorporates TLS 1.3 [Res18] encryption, thereby reducing latency during connection establishment in comparison to the combination of TCP and TLS. Developing QUIC on top of UDP and the integration of encryption enhances the robustness of the protocol, making it more resistant to existing Internet middleboxes and ossification. QUIC is generally[1] implemented as a user-space library, providing greater flexibility and enabling faster evolution of the protocol compared to TCP, which is implemented in kernel space. Another notable feature of QUIC is its ability to perform multiplexing, similar to HTTP/2 [BPT15]. Within a single QUIC connection, multiple QUIC streams can be established. Despite its advantages, HTTP/2 faces a major problem: Head-of-Line (HoL) blocking. This problem arises because HTTP/2 relies on a single TCP connection to

[1] Although QUIC was originally conceived as a user-space protocol, some libraries such as MsQuic [Mic19] provide QUIC support in the Windows kernel.

handle all its HTTP/2 streams. When a packet is lost in one HTTP/2 stream, data transmission in the other streams is interrupted until the lost packet is retransmitted and received. As a result, data destined for the application has to wait for retransmission, resulting in delays.

QUIC, on the other hand, eliminates the problem of HoL blocking. It achieves this by implementing per-flow congestion control mechanisms, rather than applying congestion control to the entire QUIC connection. With this approach, if a packet is lost in one stream, only that specific stream is affected, allowing data in other streams to continue to be transmitted. By solving the HoL blocking problem, the stream multiplexing functionality of QUIC is an improved version of the one used in HTTP/2.

QUIC also tackles the problem of NAT rebinding. In the TCP protocol, a connection is identified by its 4-tuple $(\mathtt{IP}_{\mathtt{src}}, \mathtt{IP}_{\mathtt{dst}}, \mathtt{Port}_{\mathtt{src}}, \mathtt{Port}_{\mathtt{dst}})$. However, when a NAT device [HS99] decides to modify its mapping table, the 4-tuple observed by the two endpoints may change, resulting in ambiguity in the association of network segments with the corresponding TCP flow. With QUIC, this problem is solved by eliminating the 4-tuple dependency and using a "Connection_ID" field instead. This field enables QUIC to determine the appropriate connection for data processing, thus avoiding the complications associated with NAT rebinding.

5.2 Motivations

Internet routing protocols are a key component of the Internet control plane. Using these protocols, routers exchange information that enables them to compute their routing tables. On today's Internet, BGP [RHL06; WMS04] is the protocol used to exchange routes between ASes.

From a security viewpoint, however, BGP has severe limitations. The original version of BGP-4 [RL95] did not discuss security issues. Over the years, basic security features were added to the protocol. The standard technique to secure the exchange of BGP messages between peers is to agree on a shared password and use TCP-MD5 [Hef98] or TCP-AO [TBM10] to authenticate each TCP packet with a hash. While some vendors allow using IPSec to protect BGP sessions [Jun22], this technique is not frequently deployed [FLM08]. BGP can be targeted by different types of attacks [Mur06]. In particular, a BGP session that does not use authentication can be shutdown by attackers who are able to send spoofed TCP RST [ZMW07]. Operators prevent these attacks on eBGP sessions by forcing the TTL to 255 [Pig+07]. For iBGP sessions, they usually rely on packet filters to block spoofed packets targeted at their routers. More secure BGP extensions have been proposed [LS17], but they are far from being deployed despite recent progress on the deployment of the Resource PKI [Chu+19].

During the last years, secure transport protocols made a lot of progress with the standardization of TLS 1.3 [Res18] and QUIC [IT21]. While QUIC was designed to improve web performance [Lan+17], it starts to be used by other applications such as DNS [Kos+22; HDM22] or live media transport [Cur+23]. In this Chapter, we explore the benefits that QUIC could bring to routing protocols. For this, we extend the BIRD routing daemon to exchange BGP messages over QUIC. QUIC obviously provides stronger security mechanisms than TCP. We test different use cases with our prototype implementation. We show that surprisingly the main benefits come from the ability of using certificates. Thanks to these certificates, routers can safely accept new BGP sessions from distant routers. This enables very interesting use cases that we discuss in this chapter.

5.3 QUIC for routing protocols

QUIC is a protocol that spans several layers. It integrates transport mechanisms over UDP, authentication and data encryption with TLS1.3 [Res18] and advanced transport features such as the support for multiple streams. In this section, we present the key features of QUIC and their applicability to BGP starting from the transport features. We also discuss possible improvements to QUIC tailored to routing applications.

5.3.1 QUIC transport features

QUIC is implemented as a user space library. From an operational point of view, this allows router vendors to easily integrate the transport protocol into their software with limited dependencies on the operating system. There are many open-source and mature QUIC implementations [Clo; Mic19; IET] using different programming languages.

Fragmentation and packet reordering. QUIC being a transport protocol, it efficiently handles data fragmentation and packet reordering. Offloading the entire transport of the routing protocols greatly simplifies the protocol's specification and allows one to focus on routing and not on transporting variable length messages.

Stream multiplexing. QUIC supports multiplexing by creating parallel streams that carry messages. This feature implies that it is possible to prioritize some data over others. In an ISP network, not all IP prefixes are equal. Some prefixes such as those corresponding to DNS resolvers, CDN servers or popular destinations carry much more data than many distant prefixes. If a route change affects one of these prefixes and others, the routing message carrying

the change of the important prefix could be delayed by the transmission of other messages. Router vendors already support the prioritization of the insertion of specific prefixes in the FIB. With QUIC, information about these critical prefixes could be placed in separate streams to ensure that they are not delayed, for improved BGP convergence [Bre+20]. A recent IETF draft discusses the usage of one QUIC stream per address family in BGP [Ret+23].

QUIC provides efficient per flow control and congestion control mechanisms. Thanks to QUIC's flow control capabilities, it can enable certain streams to be given priority over others. For example, in the context of BGP, one might consider a dedicated flow for BGP control messages (e.g., such as BGP notifications and OPEN messages) and separate streams for BGP route updates. By assigning a higher priority to the control message stream, we can emphasize its criticality relative to other flows. QUIC facilitates the prioritization of streams according to their importance, providing the application with a way to accomplish this task.

QUIC connections can migrate. An important feature of QUIC is its ability to survive to IP address changes. This feature was designed for mobile devices equipped with Wi-Fi and cellular interfaces. On such devices, it is essential to preserve the connection even if an interface fails. QUIC allows such a device to migrate its connections from one network interface to another without disruption. This feature can also be useful for routers that have multiple interfaces. Since TCP does not support connection migration, today's best current practice is to advertise a loopback interface on each router and use this interface to establish the iBGP sessions. With QUIC, these loopback interfaces are not required anymore since a QUIC connection can survive as long as one of the router's interfaces remains active. Currently, QUIC only allows the client (i.e., the router that initiated the QUIC connection) to migrate [IT21]. Ongoing work on Multipath QUIC [Liu+23; PB22] is the first step to enable the two peers involved in the same QUIC connection, i.e., either the client or the server, to migrate connections. Exposing additional IP addresses to connect to the server is a first requirement. But the client must do the same, and a coordination system with a new QUIC extension must be set up to launch the migration.

Native support to check that sessions remain alive. To support fast reroute, modern routers use BFD [KW10] to detect link and router failures. Since BFD can be implemented on the line cards, it will still be used by routing protocols running over QUIC. When BFD is used, the BGP Keepalive messages allow verifying that the routing process is still correctly working.

QUIC already supports the PING frame to maintain a QUIC connection alive. However, the QUIC design allows more flexibility. New QUIC frames can be added to easily support future extensions. Such an enhanced PING frame could periodically prompt the routing daemon to respond to requests from the QUIC stack. If the daemon responds, QUIC generates an enhanced reply PING frame and sends it to the remote end. The frame also functions as an indicator of whether the protocol is actively responding and continuing its usual operation. Hence, application keepalives can be offloaded to the QUIC stack, which simplifies the design of the routing protocols.

5.3.2 QUIC improves Security

Router authentication. TLS supports two types of certificates: the classical server certificates used by web servers and the client certificates. Client certificates are typically used to support enterprise VPN services using TLS or DTLS. Routing over QUIC leverages these certificates to provide a strong authentication of all routers. Each router is configured with a unique certificate assigned by the network operator. Thanks to these certificates, routers can easily authenticate the BGP sessions established by their peers. Coupled with QUIC, this provides much stronger authentication than the hash-based techniques used by BGP. Note that the certificates used by the routers do not necessarily need to be provided by an independent certification authority. For iBGP over QUIC, the certificates are used by routers managed by the network operator. For eBGP over QUIC, it would be useful to have a public certification authority, possibly related to the RPKI, that certifies the AS number authorized to announce IP prefixes [LK12]. A resource owner could register certificates for its routers. Then, a router belonging to ASx could use its certificate to confirm its identity when establishing an eBGP session with a peer.

Countering BGP attacks with spoofed packets. BGP is vulnerable to attacks where spoofed RST, FIN or malicious data are injected to force the termination of a TCP connection [Mur06]. With BGP over QUIC, these attacks become almost impossible. To inject a packet in a QUIC connection, an attacker needs to predict the current QUIC connection identifier. This is feasible if the attacker is able to observe the connection packets, but peers can mitigate the attack by regularly changing their connection identifiers in the encrypted payload [IT21]. A packet with an invalid connection identifier is simply dropped. The next step for the attacker is to predict the packet number which is part of the encrypted header. An attacker cannot easily determine this number even by capturing packets. To inject data the attacker would then need to predict the encryption and authentication keys that were negotiated using Diffie-Hellman during connection establishment. Injecting QUIC packets

is thus much more difficult than injecting TCP packets. Furthermore, routers can still use techniques such as access lists and GTSM [Pig+07] to restrict the packets that reach routers.

A router does not become a public QUIC server. Despite using QUIC to support BGP, a router does not become a public QUIC server that accepts any connection. Current routers are protected using access lists and rate limits. The same will apply to QUIC based routers. In addition, QUIC uses a connection identifier to identify a QUIC session between two endpoints. This identifier is arbitrarily chosen by both parties. To avoid potential attacks overloading QUIC with connection requests, the connection identifier could be generated using a "secret" known only to the routers. When it receives a connection request, the QUIC server can check whether the connection ID was generated using the secret before processing the request. If the connection ID is not based on the secret, the request is rejected.

5.4 Prototyping Routing over QUIC

In this section, we present an overview of Routing over QUIC (RoQ) and our prototype implementation. RoQ's objective is to replace the transport protocol used in routing protocols with QUIC. In this section, we describe how we integrate QUIC into BGP, evaluate the performance of the transport layer replacement, and present the experimental setup used to evaluate the use cases discussed in subsequent sections.

5.4.1 Architecture

Routing daemons use the system API of the operating system to access the corresponding transport protocol functions. To program an application that needs to access a transport protocol, the POSIX/BSD socket API is commonly used as the primary abstraction. However, TCP, IP, and Ethernet are protocols that are usually implemented in the kernel, whereas QUIC is a user-space protocol that is typically provided as a library. This difference in implementation implies that the API offered by different QUIC libraries to the application may vary from one implementation to another, in contrast to other transport protocols.

Our goal is to reduce the dependence of routing protocols to their transport layer. We demonstrate the feasibility of this approach by designing a socket API that abstracts the primary features of transport protocols. As a result, routing protocols can more easily switch between different transport protocols without the need for significant modifications. We believe that this will help improve the performance and reliability of routing protocols in the long run.

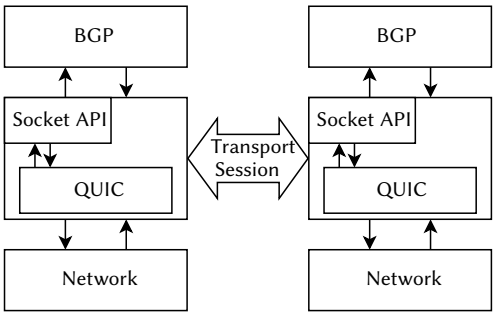


Figure 5.2: The routing protocol uses the QUIC stack to transport their routing messages.

Since the transport layer is separated from the routing protocol's base code, maintenance becomes easier and allows a faster evolution. The IETF TAPS working group also works on alternative transport architectures [Pau+23].

Figure 5.2 illustrates the architecture of our prototype. The routing protocols, like BGP, communicate directly with the transport layer via the socket API. For this chapter, we focus on QUIC, but other protocols exposing the same characteristics, as described in Section 5.3, could similarly be integrated. The transport layer establishes a transport session to facilitate the exchange of messages between two routing protocol instances. Ultimately, BGP should be considered as a mechanism for synchronizing routes between numerous routers, with no transport layer awareness.

Our QUIC socket API is composed of 4737 Lines of Code (LoC) and supports `picoquic` [Hui22]. `picoquic` is a QUIC implementation that includes a range of diverse features that are defined by the QUIC standard [IT21], as well as extensions that are currently being discussed within the IETF. This includes support for multipath [Liu+23]. Given its extensive range of features, `picoquic` presents a suitable choice for researchers and developers who want to experiment with the latest extensions of the QUIC protocol.

We created the socket API and integrated it into the BIRD [CZN20] routing stack, making it available for the routing protocols supported by BIRD. BIRD runs on various operating systems such as BSD variants and Linux. To achieve cross-platform compatibility, BIRD developed an abstraction layer on the socket interface provided by the OS. This abstraction enables the creation of different socket types, including TCP, UDP, and Raw IP. Using our QUIC socket API is straightforward; it only requires adding a few lines of code to BIRD since it closely resembles the BSD socket interface. We had to adapt some of BIRD's socket functions, such as `open`, `close`, `read`, `write`, etc. The decision to use a socket API for QUIC aimed at minimizing modifications to the BIRD code base. The total modification required in BIRD to support our

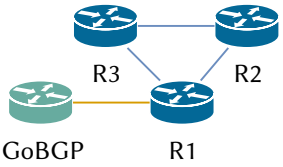


Figure 5.3: Test topology to evaluate BGP over QUIC.

QUIC socket API consisted of only 759 LoCs.

5.4.2 Performance considerations

Transitioning from an insecure protocol to QUIC introduces new security features, which will be discussed in Section 5.5. However, this may affect the performance of the transport layer since it adds the overhead of encrypting and decrypting data. Also, it is worth noting that our socket API is implemented as an overlay on top of `picoquic`, which may introduce additional delays. These delays may impact the overall performance of the transport, but they are not directly related to QUIC. It is important to keep in mind that our implementation is primarily focused on the security aspects of the transport, rather than performance optimizations. Other studies have focused specifically on the performance of QUIC and may provide more in-depth analysis on its performance characteristics [Shr+21; YB21; Jae+23].

BGP over QUIC. To assess the performance impact of the new transport stack on BGP, we create a network topology with four routers: R1, R2, R3, and GoBGP. As depicted in Figure 5.3, R1, R2, and R3 are connected in a triangle, while GoBGP is connected to R1. The GoBGP implementation [Fuj+23] is fed with a full routing table from a RIPE RIS snapshot of `rrc00`[2] dated from 23 November 2022 at 8 a.m. and establishes a BGP over TCP session with R1. In total, 970k IPv4 routes and 171k IPv6 routes are injected on R1. The GoBGP instance was run on a different server than the three Rx routers. When R1 receives the routes from the GoBGP node, it starts to propagate them to R2, then in turn R2 sends the routes to R3. Finally, R3 reannounces the routes to R1. R1 monitors all BGP update messages received from R3. R1 is configured to not send announcements to R3. Routers R1, R2, and R3 are configured to establish a QUIC or TCP transport session depending on the experiment. We measure the time taken for each prefix to be fully propagated, i.e., from router R1 to R2, R3, and then back to R1. The results, depicted in Figure 5.4, show that using QUIC does not significantly delay the propagation of BGP routes compared to TCP, despite the added security benefits of QUIC.

[2]See: `https://data.ris.ripe.net/rrc00/`

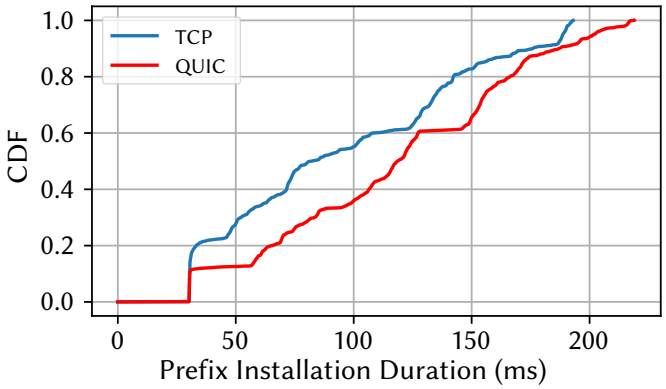


Figure 5.4: Time to install each prefix in the RIB of all BGP routers in the network.

5.4.3 Experimental evaluation setup

Realistically evaluating implementations of routing protocols at scale is not an easy task. Previous work showed that simulation is not yet practicable at large scale [RB22]. Each evaluation discussed in this document is performed on network topologies emulated on a cluster of four servers.

Hardware setup. Two servers embed dual `Intel(R) Xeon(R) Gold 5317 CPU @ 3.00GHz` (12 cores each) and the two remaining embed dual `Intel(R) Xeon(R) CPU E5-2687W v3 @ 3.10GHz` (10 cores each). They are all connected together via a gigabit switch.

Software setup. Each server runs Ubuntu 22.04. Hyperthreading is disabled and all the CPU cores except one per server are isolated[3]. Hence, the kernel runs on a single core while the others are dedicated to our emulated network nodes. The network nodes are modeled as network namespaces (`netns`) associated with two dedicated physical cores. We ensure that each pair of cores lies on the same socket to reduce the risk of contention due to cache accesses. Routing processes are launched in their associated `netns` and pinned to a first dedicated physical core. As our prototypes also embed a QUIC server with its own I/O loop, the server process is pinned to the second physical core. With that model, each server emulates a maximum of nine routers. The links are emulated either as virtual Ethernet interfaces pairs if the adjacent nodes are on the same server, or as a dedicated VLAN when the adjacent nodes are located on different servers. The VLANs are enforced at the interface level on

[3]This setup is performed with the Linux kernel configuration argument `isolcpus`.

the switch. Hence, the packets are not broadcasted on the bridge. Each virtual link is configured with a one-way delay of 10 ms.

5.5 BGP over QUIC

In this section, we present some benefits of using BGP over QUIC. We show (Section 5.5.1) that by embedding additional information in the X.509 certificates, network operators can autoconfigure services they propose at client-side. Hence, this solves manual BGP configuration burden and failures resulting from erroneous configurations. This improved flexibility is illustrated with two use cases. We first leverage secure connections offered by QUIC to dynamically contact routers in remote ASes (Section 5.5.2). We show that an edge router can establish an eBGP session without being manually configured by a network operator, to respond to an unusual network event. The second use case (Section 5.5.3) is related to improved BGP blackholing service. Thanks to the secure remote connection ability of QUIC, a client AS can establish a multi-hop BGP session towards the remote blackholing service.

These use cases leverage QUIC for its ability to secure the BGP session. The end-points are authenticated, and the communication is encrypted. The authentication relies on certificates which we enrich to exchange session information. These certificates are complementary to the resources protected with the RPKI [LK12]. The former contributes to the security of the exchange while the latter concerns the security of the resources, the origin ASes and prefixes [LKK12], advertised in BGP.

5.5.1 Dynamic reconfiguration of eBGP sessions

An external BGP session (eBGP) is a session that is configured between two routers belonging to two different ASes. These routers can be connected through a directed link or attached to the same Internet eXchange Point (IXP). There are two very common types of eBGP sessions: (i) the sessions between a customer and its provider and (ii) the sessions between shared-cost peers. To activate an eBGP session, both network operators need to manually configure the peering routers with the AS number of the peer, its IP address, the export and import filters that need to be applied, ... The configuration of these eBGP sessions can be complex and a small error in one of the commands or its parameters can result in what operator calls a *fat-finger* error. For example, if a network operator types in the wrong IP prefix, they can announce one that belongs to another network. Some of these errors are fixed quickly, others last much longer, and some have had a huge impact on the global Internet [MWA02; Tes+19; Nem+21].

In this section, we explore another approach to configure these eBGP

sessions. Instead of relying exclusively on manual configurations, we leverage the X.509 certificates that the routers exchange during the establishment of a BGP over QUIC session. An X.509 certificate [Boe+08] used by HTTPS binds a public key to a DNS name. This binding is signed by a public certification authority. The X.509 certificate format [Boe+08] is flexible and companies can deploy their own Public Key Infrastructure. To show the potential of this approach, we implemented a prototype, as depicted in Figure 5.5. Our prototype is written in 602 LoCs. We now discuss three use cases related to three different contexts of use that can be exploited with our prototype.

eBGP prefix filters. On an eBGP session with a customer AS, network operators usually configure an import filter [Ste06] that verifies that the customer only announces its own prefixes or the prefixes of its own customers. Many operators configure these filters based on the information stored in the Internet Routing Registry databases. However, adding these filters on a per eBGP session basis on each router can be cumbersome. Instead, we extend the X.509 certificate with one field that lists all the prefixes that the owner of the certificate is authorized to advertise. This certificate is issued by the provider AS, e.g., through a web portal managed by the network operator. It is signed by the certification authority of the provider AS.

To initiate an eBGP session with its provider, the client needs to use this certificate. Upon validation of the certificate, the provider router extracts the filter list and automatically configures its import filter. This is illustrated in Figure 5.5. When our BGP over QUIC prototype accepts a BGP over QUIC session, it validates the certificate and exports the contents of selected fields through a Unix socket to a script that dynamically reconfigures the import filters of the router and reloads the configuration. If a new prefix is assigned to the customer, it needs to ask a new certificate from its provider and restarts the BGP over QUIC session using the new certificate. Thanks to the BGP graceful restart mechanism [Rek+07], it is worth noting that restarting the BGP session and changing the QUIC certificate can be done without any impact on packet forwarding.

eBGP open-peerings. This use case applies to the shared-cost eBGP sessions and specifically to the network operators that have open peering policies [LDD14; Lod+14]. These ASes are present at Internet eXchange Points and accept to peer with any other AS. Usually, these operators ask their peer to respect some requirements before configuring the eBGP session. For example, they ask the peer to not modify the Next-Hop attribute, to only send traffic destined to the AS, to not point a default route to the AS or not leak the route to

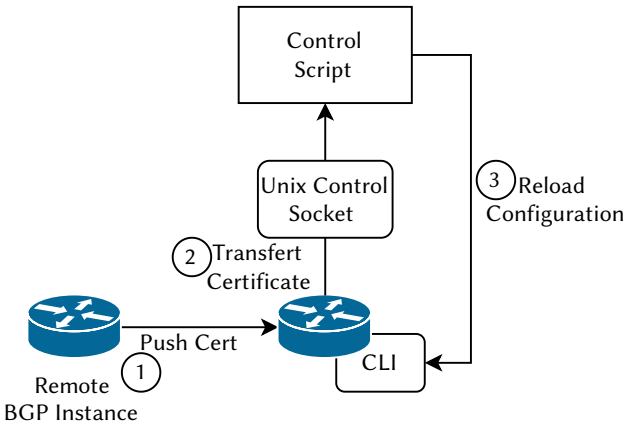


Figure 5.5: With BGP over QUIC, a router can automatically configure a BGP session based on information contained in the X.509 certificate of the peer.

other ASes[4]. Currently, the open peering AS cannot verify these requirements at BGP session establishment. The best they can do is monitoring the traffic a posteriori.

To eliminate reliance on the fragile trust system between the two ASes, the AS that accepts open peerings has the option to deploy a website. This website would enable peers to request a peering session with the AS. They would fill in certain fields such as the prefixes they wish to advertise, as well as other options for establishing the peering.

When the request is validated, the peer receives an X.509 certificate that authorizes it to establish a peering with the Open Peer AS (OP). The certificate contains two routing-related parts. The first is the configuration that the peer router must perform in its BGP router to establish a session with the open AS. The second part is an encrypted field that contains the configuration of the OP's router that will be set once the BGP session is established. The field is encrypted because it may contain sensitive data about the OP's internal AS network. The content of both encrypted and unencrypted fields can contain various information such as the IP prefixes that will be advertised by the peer, the export filter, the maximum number of prefixes to be advertised, etc. The X.509 format is flexible enough to allow the addition of these new routing-related fields.

During the BGP establishment process, the peer BGP router reads the configuration that must be applied. When the session is correctly configured, the peer initiates the QUIC connection by presenting its X.509 certificate to the OP. Upon reception, the OP router reads and decrypts the field reserved

[4]See examples of those requirements here: `https://openpeering.nl/`

for them, applies the configuration and accepts the QUIC session.

BGP communities. This use case is about improvements of the traffic engineering features offered by a provider to its client ASes. The best current practice to allow a customer AS to request traffic engineering features from its provider is to rely on BGP communities [LCT96]. These are opaque BGP attributes that are only understood by the AS that defined their semantics. They are used for various functions such as traffic engineering, limiting the propagation of routes through certain regions of the world, etc. [DB08; KBS21]. In addition, some operators publicly disclose how to use BGP communities to influence routing[5]. Unfortunately, BGP communities can also be used for malicious purposes to trigger unexpected blackholing or perform DDoS [Str+18; MP19; Bir+19].

We leverage the X.509 certificates to provide an alternative to these BGP communities. The X.509 certificate used by BGP over QUIC can also contain commands that configure the associated BGP session. A provider AS can create a web portal where customers authenticate that allows them to request specific traffic engineering configurations for some of their prefixes (e.g., set different `local-pref` values for different prefixes, restrict the distribution of some prefixes to parts of the provider network or its peers, etc.). Automated validation of the requested actions can be applied before issuing the certificate. This approach is more secure than the current BGP communities which are not authenticated. The BGP commands that are placed in the X.509 certificates are always provided by the network operator and not its customer. For more specialized techniques, a network engineer could manually create the required commands. The commands placed in the certificates can be validated by verification tools [Bec+17a] before being included in the certificate.

Security Considerations It should be noted that the generated certificate may potentially contain errors, whether intentionally or unintentionally introduced by the operator. However, for the purposes of this work, we assume that the certificate contains no errors and that it has been subjected to rigorous testing before being deployed in a real environment.

In a future iteration of this work, it could be a better approach to develop a system that automates the generation of certificate configurations with minimal user intervention, incorporating a validation mechanism that uses specialized tools such as Snowcap [SBV21], Chameleon [Sch+23] or BatFish [Fog+15]. These automated approaches would help improve the accuracy and reliability of certificate configurations, reducing the risk of errors, transient

[5] See e.g., NTT BGP Communities: `https://www.gin.ntt.net/support-center/policies-procedures/routing/`

states or policy violations in the deployment process.

5.5.2 On-demand BGP over QUIC sessions

BGP was designed under the assumption that eBGP sessions are configured over a physical link and remain up as long as the link remains active. However, there are many situations where it could be useful to create and terminate an eBGP session in response to traffic load or external events such as link failures.

A first example relates to the IXP networks or other carrier-neutral network facilities. While a small IXP can be composed of a simple L2 switch, large IXPs are metropolitan L2 networks that are spread across multiple sites or even multiple cities. At a high level, an IXP network can be thought of as a large L2 network where ASes have one or more points of presence, either at the same site or at different sites within the IXP network. Many IXPs also offer additional services to their customers, from route servers [Jas+16] to the ability to create VLANs or any other techniques that enable the creation of a virtual L2 network across the IXP network[6]. Thanks to these techniques, ISPs can create direct links over the IXP infrastructure to their peers, but also some of their providers and customers. These L2 networks are also used to support remote peering services [Gio+20; Cas+14]. Figure 5.6 shows an example of such configuration. The IXP network attaches routers from a pair of ASes to the same virtual L2 Network. The stub router will usually not establish an eBGP session with every other router of its provider since this service is often charged per eBGP session and in function of the bandwidth used. However, there are situations where it is useful for the stub router to create additional eBGP sessions with its provider. A first example is when the provider needs to perform some maintenance on its router. In this case, the stub router could establish an alternate eBGP session with another router of the same provider over the IXP network. A second example is when the traffic on the stub router increases. Instead of simply increasing the bandwidth to the provider, it could be useful to create an additional session with another router of the same provider. A third example are customer networks that need additional capacity for small periods of time. For example, a stadium could add more eBGP peering links when the stadium hosts an event and disable them after the event [ER13]. With the X.509 certificates, a provider router can safely accept a new eBGP session from another router of a customer AS since this customer is authenticated by the certificates.

[6] See Equinix VLANs: `https://docs.equinix.com/en-us/Content/Interconnection/IX/IX-bilateral-peering.htm`, BNIX: `https://www.bnix.net/en/services/private-vlan` or DEC-IX VirtualPNI: `https://www.de-cix.net/en/services/virtualpni`

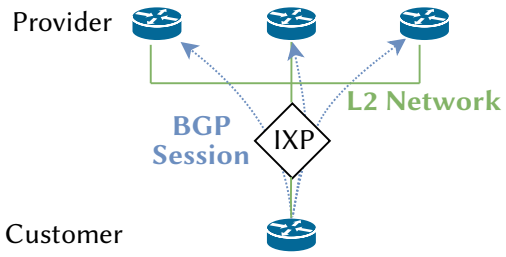


Figure 5.6: The IXP network provides VLANs between pairs of ASes so that they can directly create peering sessions.

5.5.3 Improved Blackholing service

Reconfiguring client routers is not the only service enabled by X.509 certificates. X.509 certificates also allow improving the Remote Triggered Black Hole (RTBH) service [Tur04]. RTBH is a method that consists in altering BGP tables to counter Distributed Denial of Service (DDoS) attacks targeting client networks. The undesired traffic is simply dropped before reaching the target hosts. Unlike traditional reactive methods that filter traffic after it has reached the target network or host, RTBH drops traffic at the edge of the provider network. This filtering is built in two phases [Cis05]: a preparation phase where the routers are configured with a route to be used for blackholing, and the announcement to redirect the traffic to that route. This is when blackholing takes place. First, a static `null` route is configured on one or several of the provider's routers called the blackhole routers. The packets towards this `null` route or having this `null` route as their nexthop are discarded.

In the blackholing step, the announcement may come from inside the network or cross AS boundaries. Inside the provider's network, a BGP session is established with a special router, called trigger router. When a DDoS attack occurs, the customer router sends to the trigger router a BGP announcement containing the address of the attack target and the destination of the `null` route as its nexthop. This route is distributed in the provider's network only. When a router of the provider's network receives a packet destined to the target address, it automatically discards it. This allows the targeted network or host to continue normal operations while the attack traffic is dropped as closely as possible from its source.

It is also possible to communicate the need for blackholing across an ISP border. Here, a BGP community triggers the installation of the `null` route. Some operators or transit providers provide this blackholing service for their customers, as illustrated in Figure 5.7. When a client AS detects a DDoS attack and wishes to block all packets coming from this provider towards the target address under attack, it informs the provider AS by re-announcing the route

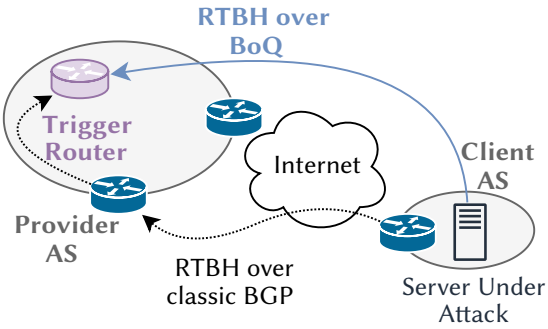


Figure 5.7: With traditional RTBH, blackholing information follows the BGP paths up to a trigger router. With BGP over QUIC, a customer AS can directly request a remote blackhole from distant ISPs.

with a special BGP community. This solution is widely used, but researchers have shown that it suffers from several problems [Naw+19; MP19]. First, BGP routes can be inadvertently or maliciously manipulated [AZV17; Cho+19]. Second, a third party AS can trigger such remote blackholing for particular prefixes [Str+18].

By leveraging the certificates supported by BGP over QUIC, we propose a new way for service providers to support RTBH. Instead of extracting the RTBH information from the received BGP communities, an ISP could provide RTBH for any AS or the customers of its customers. Consider a Tier-1 ISP that carries a lot of traffic and wants to support RTBH. This ISP would first issue certificates to the ISPs that subscribe to its RTBH service. This certificate contains two information. First, the certificate lists the IP prefixes that the customer of the RTBH service owns. Second, it contains the addresses of the trigger router in the Tier-1 ISP.

When a customer network detects an attack that targets one of its IP addresses, it can install an RTBH blackhole in one of its subscribed Tier-1 RTBH services. To install the blackhole, the customer ISP dynamically establishes a BGP over QUIC session with one of the trigger router of the Tier-1 ISP. This process can be automated on the customer side since the certificate contains the IP addresses of the Tier-1 trigger routers. When the trigger router accepts the BGP over QUIC session, it verifies the certificate and installs an import filter that matches only the customer's prefixes. The customer then announces the attacked IP address over the BGP over QUIC session. The trigger router ensures that the address belongs to the customer's addresses listed in the certificate and installs the blackhole. The blackhole lasts until either the customer withdraws the announcement on the BGP over QUIC session or the session stops.

To demonstrate this enhanced RTBH service, we set up a small network

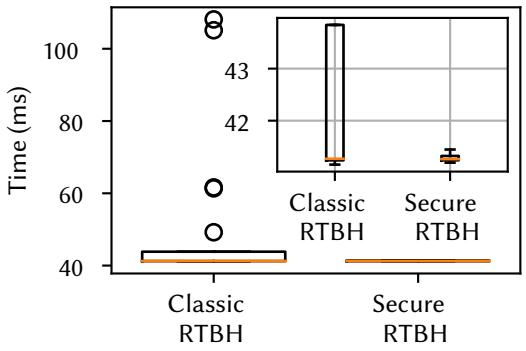


Figure 5.8: Time to setup a blackhole for a prefix selected by a client.

depicted in Figure 5.7. When the operator detects a DDoS attack toward one of its servers, it requests a router to announce the server's address to be blackholed. During this operation, several intermediate BGP nodes can be present on the path between the trigger router of the provider and the router requesting a blackhole. The advertisement may cross multiple ASes. With BGP over QUIC RTBH version, the client AS directly establishes a QUIC session with the trigger router. This prevents any manipulation of the announcement [MP19] and avoids the need to propagate the blackholed route through several intermediate BGP speakers, hence minimizing the propagation time of the blackhole request.

Figure 5.8 shows our measurements of the duration required to enable the blackholing service. That is, the delay from the instant the server announces the blackholing to the moment the provider AS has enabled the `null` routes on its edge router. We run the experiment 30 times for both the classical and the QUIC-based RTBH. With classical RTBH, the median time to setup the blackhole is 41.27 ms, while with our QUIC-based RTBH, the setup time is 41.26 ms. One can note that QUIC-based RTBH is more stable than the classical RTBH. Indeed, the variance and the outliers are reflecting the internal processing of intermediate nodes. In the global Internet, this delay can be much longer due to timers like MRAI [GB19; TUD07; FSR11]. With QUIC-based RTBH, the propagation time is stable as the route is only announced to the trigger router. Furthermore, the blackholing request is protected by the QUIC session and cannot be modified.

BGP over QUIC opens a more secure way to support remote triggering black holing (RTBH). This enhanced RTBH could be widely supported by large ISPs to provide better reactions to the unfortunately frequent denial of service attacks.

5.6 Related Work

The use of QUIC in BGP. The IETF has started to work on using QUIC to carry BGP messages [Ret+23; RQT22]. Our implementation of BGP over QUIC is partially aligned with this effort. At the time of writing, after discussion with the draft authors and to the best of our knowledge, we are the first to have an implementation of BoQ. The use cases described in this document go beyond the current IETF discussions.

BGP over QUIC secures the exchange of messages and the certificates authenticate the peers. Our BGP over QUIC prototype secures a BGP session, whether it is a single hop or multi hop session. This means that an external attacker cannot inject messages without knowing the encryption key used for the session. Another effort has also focused on encrypting BGP messages by directly modifying the protocol [SG96]. However, BoQ cannot protect against the injection of false information by a malicious AS, such as changing the AS path or the origin of an IP prefix. Other extensions, such as BGPSec [LS17] and RPKI ROA [HM12], provide protection against routing modifications by ASes. Several approaches have been proposed to address this problem, including SPV [HPS04], soBGP [Whi03], S-BGP [KLS00], PGBGP [KFR06], and IRV [Goo+03].

Secure tunnel protocols. Securing the transport of routing messages can be also made with IPSec [SK05; Ken05a; Ken05b] or any other secured tunneling protocol such as Wireguard [Don17] or OpenVPN [Fei06]. These protocols encapsulate routing messages in an encrypted payload. In the case of IPSec, the Internet Key Exchange (IKE) [Kau05] protocol (i) authenticates the two parties with Pre Shared Keys (PSKs) or X.509 certificates and (ii) negotiate the keys used to encrypt the payload. The routers can then securely send their data as the encrypted tunnel provides authenticity, message replay prevention, confidentiality and integrity. QUIC provides the same security guarantees but reduces the configuration overhead, as there is no need to have an extra protocol to encrypt and authenticate data.

5.7 Conclusion

In this Chapter, we have demonstrated the benefits that QUIC brings to BGP. A first benefit is that this protocols become less susceptible to packet injection attacks. The main benefits come from the ability to authenticate BGP sessions using X.509 certificates.

We propose an advanced usage of X.509 certificates allowing the automation of critical configurations for which human errors have significant con-

sequences. In addition, thanks to the mutual authentication enabled by such certificates we have demonstrated on-demand BGP sessions and improved remote blackholing services. We have also shown that these certificates allow to automatically and safely configure prefix filters and traffic engineering features on ISP routers.

BGP over QUIC is another first step towards modernizing the Internet routing protocols. We expect that new use cases will be developed in the future and that other Internet control plane protocols could also benefit from QUIC.

Part IV

Making a BGP data-plane “aware”

Checking the Reachability of BGP Routes Using the Dataplane

6

This chapter is largely based on the paper T. Wirtgen and O. Bonaventure. "A First Step towards Checking BGP Routes in the Dataplane". In: *Proceedings of the ACM SIGCOMM Workshop on Future of Internet Routing & Addressing*. FIRA '22. Amsterdam, Netherlands: Association for Computing Machinery, 2022, pp. 50–57. ISBN: 9781450393287. DOI: 10.1145/3527974.3545723. URL: https://doi.org/10.1145/3527974.3545723.

The last contribution of this thesis explores the interactions between the control-plane and the data-plane in routing protocols. Distributed routing protocols use control-plane information to determine the optimal route for an IP prefix. This information is then incorporated into the Routing Information Base (RIB) and propagated to the Forwarding Information Base (FIB) to enable packets to be forwarded. Each routing protocol, such as BGP, OSPF and IS-IS, uses its own mechanisms for this purpose.

In the case of BGP, a set of control-plane attributes are used to discern and differentiate routes. These attributes include factors such as AS-Path, next-hop information, route origin and other policy-related criteria. By analyzing and taking into account these attributes, BGP is able to make decisions about the best path towards a given prefix. This attribute-based approach enables BGP to support complex routing policies.

In contrast, the OSPF and IS-IS protocols take a different approach, relying on manually defined weights assigned to each link. These weights serve as metrics that contribute to the calculation of the Dijkstra algorithm. Dijkstra's algorithm computes the shortest path between routers, taking into account the sum of the weights of the links connecting them. By assigning appropriate weights to different links, network administrators can influence the routing decisions made by OSPF and IS-IS, thus shaping the flow of traffic within the network.

This thesis makes a first contribution to improving the interactions between the control-plane and the data-plane. By exploiting the information

contained in the data-plane, innovative approaches can be developed to improve routing state. The aim is to enable more dynamic routing decisions based on factors that go beyond traditional control-plane metrics or static attribute values.

Routers usually transfer information from the control-plane to the data-plane. Technologies or research results exploiting the opposite direction, i.e., from the data-plane to the control-plane are limited. Some IGP protocols, such as EIGRP, use link load and reliability to calculate weight parameters. This data is obtained from counters attached to router interfaces in the data-plane. However, the interaction between the two planes is currently limited and lacks flexibility.

Blink [Hol+19] is another example of work that merges the control and data-planes. This approach relies on TCP measurements to identify link failures and dynamically reconfigure routing to avoid sending packets over the affected link. Blink's prototype shows the effectiveness of its solution using programmable switches, demonstrating its ability to quickly redirect traffic to alternate paths.

Blink has focused on programmable switches, but not on distributed routing protocols, which are still widely used on the Internet. The integration of the control-plane and the data-plane together in distributed routing protocol allows for improved innovation in networks. We believe it would become possible to dynamically control and adjust network behavior based on real-time data-plane conditions.

In this chapter, we tackle a fragment of the problem by developing a new method to check the reachability of the routes announced by BGP. Instead of directly inserting the route into the routing table, we first check whether the route prefix is reachable in the data-plane. We ask the router to contact a given destination in the prefix sent in the BGP Update. If this destination is reachable, the route can be considered valid and thus can be added to the routing table. We validate the feasability of the approach by conducting an early evaluation in a small topology which also shows the impact on BGP convergence.

Although the work presented in this chapter addresses only a small part of the integration between the data-plane and the control-plane, it serves as a practical demonstration of the feasibility and the potential of combining them. The long-term objectives of this chapter are twofold. Firstly, to open up new perspectives in this specific field. Secondly, to provide a basis for the development of other tools and approaches to achieve routing that is fully consistent with the environment in which it operates.

The remaining of this chapter is organized as follows:

- Section 6.1 first provides motivations to secure BGP routes in the data

plane.

- Second, in Section 6.2, we describe a new architecture that can be used to check if paths are reachable.
- Then, in Section 6.3, to demonstrate that it is possible to add the intelligence of the data-plane in the control plane, we implemented a new route validation method in FRRouting, an open-source implementation of BGP.
- Fourth, we discuss the implications and new possibilities of using data-plane information to influence routing in Section 6.4.
- We finally present the related work in Section 6.5 and conclude this chapter in Section 6.6.

6.1 Motivations

BGP was designed when the Internet was a research network [Jab15]. The first BGP design did not include any security feature. The initial assumption was that network operators could be trusted. Over the years, this assumption appeared to be too optimistic. First, network operators, even trusted ones, sometimes make mistakes. This is known as the *fat-finger* problem as explained in Section 5.5.1. Second, some network operators, either maliciously or through attacks, have advertised IP prefixes that belong to other ASes [Ser+18]. Researchers have identified various forms of such BGP hijacking [Cho+19].

To cope with these difficulties, and after many debates, the Internet Engineering Task Force (IETF) finally adopted the Resource Public Key Infrastructure (RPKI) model [BA13]. In a nutshell, the RPKI uses cryptography to create certificates that bind an IP prefix to an AS number. The Regional Internet Registries publish the RPKI certificates that indicate which AS can legitimately announce a given prefix. Thanks to these certificates, when a BGP router receives a new route, it can compare the origin of the route (i.e., the last AS in the AS-Path) with the information contained in the certificate. If they match, the router considers the route as valid, and it can be used and reannounced by the router. Otherwise, the router may decide to discard the route. Researchers and network operators have studied the deployment and the usage of the RPKI during the last decade [Chu+19; Reu+18; Wäh+15].

While the RPKI is slowly getting deployed, it only allows a BGP router to verify that a received route was announced by its legitimate origin AS. The RPKI does not verify the entire AS-Path, although there are proposals to perform such validation [Coh+16; Azi+20]. In the longer term, BGPSec should

improve [HB11] the security of interdomain routing, but its deployment has not yet started.

6.2 BGP routes reachability in the dataplane

During a maintenance routine in an AS, accidental configuration errors can disrupt connectivity. Configuration errors occur every day and can impact many prefixes [MWA02]. All traffic passing through the misconfigured AS can cause losses on the affected path and thus can blackhole traffic. When BGP Updates are received from misconfigured routers, the route contained in the update is by default considered to be reachable in the data plane. Therefore, the route is a potential candidate for forwarding traffic, even if it is not reachable in the dataplane. This example raises an interesting question: **How can a router validate the reachabilty of a route received from a peer?** This is a complex question because an IP prefix may contain addresses for endusers, servers or even infrastructure such as routers. From the host viewpoint, a network prefix is reachable if the host can exchange packets with one IP address belonging to this prefix. It is important to note that simply receiving packets from sources belonging to a given IP prefix does not guarantee the reachability of this prefix since such packets could have been spoofed [Luc+19]. The host should receive a reply to the packets that it sends. Several protocols can be used to elicit such a reply from remote hosts: ICMP with `ping` or possibly `traceroute`, TCP or even application-level protocols such as DNS.

Threat Model Checking the reachability of BGP routes should address the following threat model. In a network environment where a specific AS lacks control, an adversary has authority over routers and links outside the AS's jurisdiction. The adversary can also be anyone with access to the network, who can send corrupted routing information, but also modify the behavior of the data plane.

These adversaries can employ a variety of tactics, including deploying Ethernet or IP filters in the data plane to block the forwarding of victim's packets through their network. In addition, they have the ability to engage in activities such as eavesdropping on traffic, redirecting it to other network paths or intentionally diverting it. In addition, they can modify packets in transit and even launch Distributed Denial of Service (DDoS) attacks against AS services. These DDoS attacks can potentially overwhelm AS network links and equipment, rendering them inaccessible for legitimate use.

Furthermore, the attacker can manipulate BGP routers by sending falsified information, thus deliberately blocking the victim's traffic. Even in cases where the attacker does not have direct control of a router, he can harm a target AS router by forging BGP packets. To limit this latter type of attack,

security measures already counter it. These measures include the use of techniques such as GTSM [Pig+07] and the implementation of protocols such as TCP-AO [TBM10] or TCP-MD5 [Hef98] to reinforce the security of BGP communications. These measures reinforce the integrity of BGP messages and protect the routing infrastructure against unauthorized injections.

The motivation for such actions by attackers may be economic or political, aiming for example to prevent an autonomous system from accessing specific services hosted elsewhere in the network [AP15].

The goal of our solution which will be described in this section, is to improve the security of packet routing within the network, while ensuring that control plane information remains usable in the data plane. In brief, our method guarantees end-to-end connectivity. If a BGP route is announced in the control-plane then it is reachable in the data-plane.

Based on this property, our solution defends against several types of attacks, including blackhole attacks launched by a third-party AS or a compromised router. These attacks block the forwarding along its intended path, preventing other ASes from accessing the network. Our solution also provides protection against equipment failures and Distributed Denial of Service (DDoS) attacks, which can disrupt the forwarding capacity of network components.

In addition, our solution can be used to prevent attacks aimed at manipulating Route Flap Damping (RFD) [VCG98] mechanisms or Minimum Route Advertisement Interval (MRAI) [RHL06][1] timers. These attacks involve removing and re-advertising routes to trigger the router's mechanism that blocks installation of the route in the RIB while it is being received and considered as the best route [SVG16]. These attacks can be prevented if the reachability check is also performed on BGP withdrawals. If a route is announced to be withdrawn but remains reachable on the data plane, the router should not remove it from its Routing Information Base (RIB). If BGP withdrawals are verified, our solution also counters attacks where a malicious AS replays or suppresses withdrawals on an advertised route [MPE18]. Furthermore, our solution provides safety against attacks where an attacker hijacks a prefix or sub-prefix, as long as they do not redirect traffic to their intended destination, commonly referred to as interception attacks.

Nevertheless, our solution does not check whether the path taken by the data plane corresponds to the AS-Path as advertised in the control plane. In other words, our approach does not prevent collusion attacks (also called wormhole attacks) [Li+15; LHZ14] or any other form of attack aimed at diverting the forwarding of packets from the route initially prescribed by the control plane. The solution also provides no protection against an AS eavesdropping on the end-to-end reachability check. For example, if an AS knows that checks

[1] See section 9.2.1.1 of RFC4271.

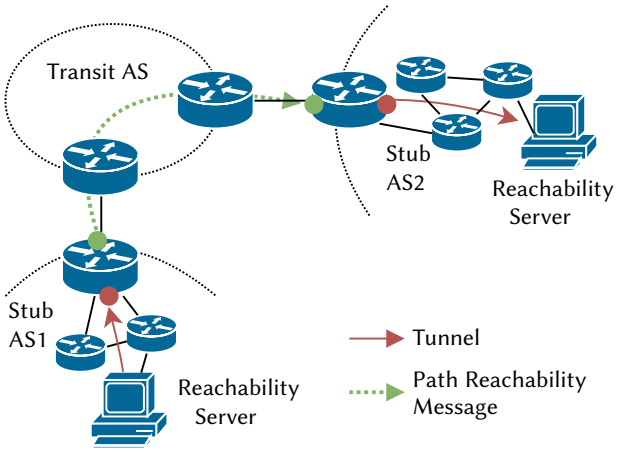


Figure 6.1: Architecture of the proposed solution.

are being performed on a particular port, it can accept the requests during the reachability check and then set up IP filters to block normal traffic.

Design of the Solution To address our problem, a first possibility to validate the reachability of a prefix is to send `ping` packets to one or more IP addresses belonging to this prefix. Researchers and network operators frequently use `ping` to verify that an IP address is reachable. However, `ping` has two important limitations. First, for security reasons, only a small fraction of the Internet hosts respond to `ping` packets. This means that finding reachable IP addresses inside a prefix can be difficult, in particular for IPv6, even if there are hit lists of *pingable* IPv4 [QHP13] and IPv6 hosts [Gas+18]. Second, `ping` is not a secure protocol and spoofed replies are possible [ABD14]. Still, `ping` has the advantage of being simple to use.

A better approach is to leverage secure protocols such as TLS [Res18]. If a client can establish a TLS session with one IP address belonging to the prefix of interest, then does this necessarily confirm that the prefix is reachable? TLS was designed to allow clients to authenticate connections with a server identified by a domain name. For this, TLS relies on X.509 certificates [Boe+08] that securely bind a domain name with the server's public key. In addition, the Subject Alternative Name (SAN) extension for X.509 [Boe+08][2] certificates also enables the generation of a TLS certificate to bind public keys to IP addresses. SAN allows trusting a specific host contained in a destination prefix. X.509 certificates can therefore be used to confirm that the prefix is reachable.

With TLS certificates, we have a way to authenticate IP addresses. Now we need to deploy a solution in the network to be able to use path reachability

[2] See section 4.2.1.6 of RFC5280.

messages. A first possibility would be to integrate a TLS server in the routers so that they can respond to TLS messages. Unfortunately, adding support for TLS certificates in routers would require modifications to router operating systems to include cryptographic mechanisms to validate TLS certificates. Some routers deployed on the Internet are old and would not simply handle the introduction of TLS. For more recent network hardware, the router itself can run Virtual Machines (VM) to host small applications [Dev20], which can be the ideal solution to include a TLS server in the router to validate paths. Instead, to support a broader range of network devices, we can opt for a decentralized solution as shown in Figure 6.1. A new type of service can be introduced to verify BGP routes, we call it the *Reachability Server* (*RS*). The network operator may decide to deploy several *RS*s to spread the verification load across its network. In addition, to limit the number of verification carried out by another network and thus avoid a possible DDoS, *RS*s could be located behind a reverse proxy, protected by a Web Application Firewall (WAF) or a classical firewall that limits the number of requests per IP prefix. Instead of contacting an existing service inside the AS such as HTTP or DNS servers, the *RS* can be added inside each AS so that routers can contact it to verify the prefix. Each time a router receives a BGP Update, it will contact the *RS* to check if the route is reachable in the dataplane. This *RS* brings several advantages. First, it reduces the load on existing services. For example, an operator would not want to have an additional load on its web server. Second, the service is dedicated only for reachability and nothing more. Strong security policies can therefore be applied to this dedicated server. However, this would require additional coordination efforts between the network and the system team, as the routing devices must interact with the infrastructure. It requires more configuration to make the system work. Finally, the *RS* can be used as a cache that can speed up BGP convergence when a router performs a cold start or when it receives a large number of updates. If the route has already been validated by the *RS*, the AS routers can query it to retrieve the information, thus avoiding recontacting the target prefix with a new reachability message. In this scenario, the *RS* is also in charge of performing the reachability verification such that it can maintain its cache.

The proposed architecture is flexible, thanks to the introduction of the *RS*, when the AS2 edge router, shown in Figure 6.1, sends a BGP Update with a prefix, AS1 has several possibilities to verify the path. Either the AS1 edge router asks its *RS* to perform the reachability check. On its side, AS2 can also choose which device will respond to AS1's reachability request. It can choose to configure its router to respond to the request or to transfer the request to the *RS*. If the router or *RS* of AS1 receives the response from whatever device, the route is considered reachable and therefore can be inserted into the routing table.

Using an external machine like the *RS* also brings disadvantages. When the BGP router receives a route to a prefix and asks to the *RS* to contact a destination within the target prefix, it has no routes to the corresponding destination. As the router waits for the reachability message, it does not install the route to its routing table. Hence, the *RS* cannot contact at all the destination. Instead, the *RS* can tunnel the request to the AS1's egress router as shown in Figure 6.1. The AS1 edge router is responsible for decapsulating the request and forwarding it to the target *RS* from the interface where it received the route.

Now that we have TLS to authenticate prefixes, we need to modify the implementations of BGP to ask them to verify the received paths before integrating them into the routing table. To have a deployable solution, it is necessary to tell BGP the destination to be contacted by prefix in a secure way. Adding a new BGP community or a new BGP attribute is not secure because the messages can be intercepted. RPKI solves this problem. RPKI has been deployed to authenticate some Internet resources such as Route Origin Authorizations [LKK12] (ROAs). As of this writing, ROAs are the only RPKI objects that are widely deployed. Other works propose adding new objects into RPKI to enhance routing security [Azi+22; SsA20; SAM22]. We can imagine extending RPKI by proposing a new RPKI object, the Route Prefix Reachability (*RPR*) as described in Listing 6.1. The syntax used in the definition of the *RPR* has been adapted from the definition of ROAs object [LKK12]. In a nutshell, this extension of X.509 certificate contains a sequence of IP prefixes and a list of IP addresses that represent the *Reachability Servers* to contact in order to validate the IP prefixes. Just like ROAs, the *RPR* object will be pushed to the global infrastructure so that all RPKI validators can use them. Since the RPKI object is cryptographically signed, the IP addresses of the *RS*s included in the certificate can be trusted. It is important to stress that the IP addresses contained in the certificate must be valid for the IP prefix. This means that the IP addresses must be included in the IP prefix. This prevents anyone from validating the path of other ASes.

When the BGP router receives a route, just like the ROAs, it can decide whether or not to contact the RPKI validator to obtain the IP addresses of the *RS*s to contact to validate the BGP route. Depending on the configuration chosen by the network operator, the router or the *RS* will make the request to check if the IP prefix is accessible in the dataplane. With RPKI and the establishment of a secure communication with the *RS*, the infrastructure provides strong guarantees to find and contact the remote *RS*. It should be noted that the RPKI should not be used to prove the identity of the *RS* [BH22]. This concern is addressed by the TLS stack with X.509 certificates, as is the case for web server authentication. The *Reachability Server* will contain another X.509 certificate that will only be provided for authentication when queried to

```
RoutePrefixReachability ::= SEQUENCE {
  pfxReachable SEQUENCE (SIZE(1..M)) OF IPAddress,
  ipAddrBlocks SEQUENCE (SIZE(1..MAX)) OF
                                    RPRAddressFamily }

RPRAddressFamily ::= SEQUENCE {
  addressFamily OCTET STRING (SIZE (2..3)),
  addresses SEQUENCE (SIZE (1..MAX)) OF RPRIPAddress }

RPRIPAddress ::= SEQUENCE {
  address IPAddress,
  maxLength INTEGER OPTIONAL }

IPAddress ::= BIT STRING
```

Listing 6.1: RPKI object related to the prefix reachability.

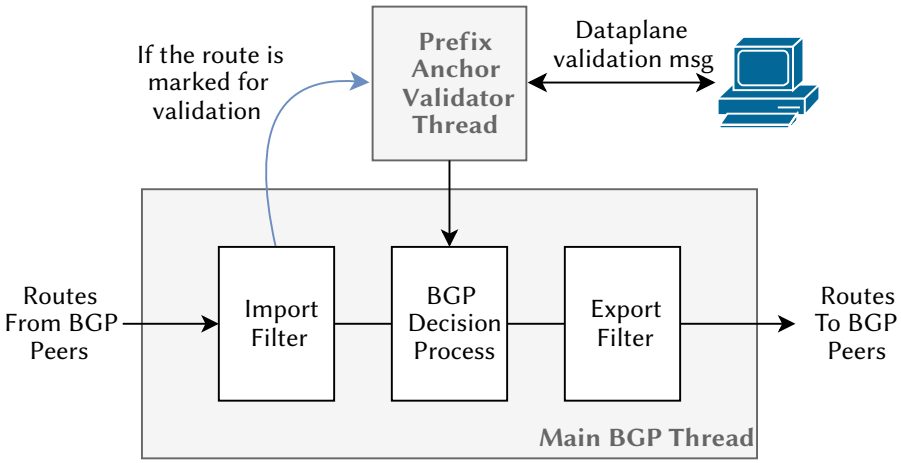


Figure 6.2: Modified Architecture of BGP to include the Prefix Anchor Validator.

validate the paths. The certificates of the *Reachability Servers* will be signed by a certification authority which could be one of the five Regional Internet Registries (RIRs) and the IANA as the root certificate authority.

6.3 A First Prototype

We developed a first prototype in the BGP module of FRRouting v8.2.2 [Fou17] with a *Prefix Anchor Verifier* (*PAV*). Our modification requires ~1.1k lines of C code. The remaining of this section explains the architecture of the *Prefix Anchor Validator* we designed to validate the routes in the data plane.

Figure 6.2 shows the general architecture of our solution. The traditional BGP workflow remains unchanged as the routes received from BGP neighbors

pass through the import filters first. If the import filters notice that a route needs to be validated, it will be passed to another thread that will perform the reachability check. As FRRouting does not parallelize the processing of BGP Updates, we could not simply perform the check inside the BGP thread that processes a BGP Update. Doing so would have dramatically slowed down the processing of BGP routes since a route that needed to be validated would have blocked the subsequent BGP routes in the same session until the completion of the task. This thread can be modified later to add the communication with an external *RS*. For simplicity, and for a first design, we deploy our solution on the router. Once the prefix is validated or not, the validator asks the main FRRouting thread to restart the BGP decision process in order to add or remove the route processed by the validator.

Our *Prefix Anchor Verifier* implementation is flexible. The current implementation offers two types of path reachability but others can easily be added in the future. The first uses a simple `ping` method. The router sends an ICMP packet and expects to receive the response within a given time period. If no response is received, the router repeats this process until the limit of retries is reached. Although using a `ping` to validate a route is not secure, we still decided to add this reachability method to our *PAV* to simplify the evaluation of our proof of concept. We also provide a second type of reachability check by using a TLS server. From a deployment perspective, this is much better than a simple `ping`. With TLS, it is possible to authenticate the server we are validating. To allow the router administrator to choose the reachability method, we modified the FRRouting CLI to allow the setting of the entire *Prefix Anchor Verifier*. Namely, the CLI allows the user to change the timeout before considering that no response has been received or the number of prefix reachability attempts.

Our implementation keeps a route that has not yet been validated in the routing table with a `NO EXPORT` [LCT96] community. This well-known community indicates that the router cannot propagate it to other neighboring ASes. When the reachability process concludes, we either remove the route if it could not be validated (rejection) or remove the `NO EXPORT` community and trigger the BGP decision process (acceptation).

When a route is received, it must be ensured that the traffic can reach the target prefix. This is done by choosing an address that is within the prefix announced by the BGP route. In order to avoid choosing a random destination, our first prototype asks each AS to add a large-community [Hei+17] that contains the destination to contact in the prefix. The next prototype will be based on RPKI certificates. But for now, we enable this information by using large-communities as shown in Figure 6.3. Since a large-community value can only contain 12 bytes, two namespace identifiers had to be reserved to represent an IPv6 address. If path reachability is enabled on the router, the

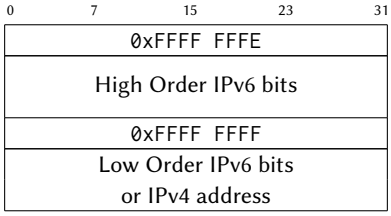


Figure 6.3: Structure of the Large-Community used for Path Validation.

```
route-map path_reachability permit 10
 match path-reachability notrequested
!
route-map path_reachability permit 20
 match path-reachability pending
 set community additive no-export
!
route-map path_reachability permit 30
 match path-reachability valid
 set community additive 65021:6
!
route-map path_reachability deny 40
```

Listing 6.2: Example of using BGP Path Validation with the FRRouting CLI.

network operator can choose to install an import or export filter to check whether a route should be verified with BGP path reachability. If the filter matches the large-community, the router triggers a parallel thread that will validate the route without disrupting the initial BGP workflow.

We have modified the FRRouting CLI to integrate the path reachability to work with route-maps [Cis15]. Listing 6.2 shows an import filter using the route-maps. In this example, the router will accept routes that are path verified (`valid`), those which do not contain the large-community values (`notrequested`) and those that are in the process of path reachability (`pending`). If the route cannot be validated, it is withdrawn (`path_reachability deny`). However, completely removing the route from the routing table is a difficult decision to take. In fact, it may happen that the route can be reached after a certain time. To avoid this, the operator can always import it into the routing table with the lowest possible local-pref to allow the router to use another valid path instead.

To show the feasibility of this approach, we use a simple network to validate BGP advertisements made by a BGP neighbor (C5) as shown in Figure 6.4. The C5 router sends a routing table of 873k routes from a RIPE RIS snapshot (June 3, 2021, at 4:15 PM). We modified this routing table to mark 2% of the routes (18k) with a large-community to instruct the C4 router to validate the

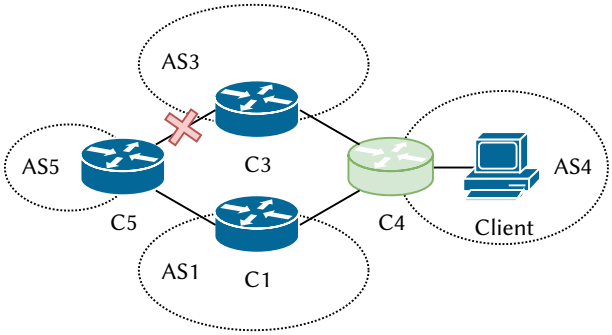


Figure 6.4: Simple network used for evaluations.

routes only for those prefixes. All routers except C4 run BIRD v2.0.9 [CZN20]. The C4 router is running our modified version of FRRouting v8.2.2 [Fou17] to enable BGP to check the reachability of a route using the dataplane. The C4 router is running an Intel® Xeon® X3440 @2.53GHz with 16 GB of RAM, Linux kernel v5.15.29 and Debian 11.

We have configured C4 using local-pref so that all routes advertised by C3 are preferred to those advertised by C1.

To emulate failures in the dataplane when C4 validates routes, we decided to install IP filters for the 18k prefixes on the C3-C5 link. These failures are used to create unreachable paths in the dataplane. This way, C4 will detect that the marked route passing through C3 should not be used. The C1-C5 link does not contain any filter. Thus, for all the routes that need to be checked, C4 will decide to go through C1 since the other route does not let path reachability messages through. Figure 6.5 shows the proportion of routes that pass through C1. At the beginning, no route goes through C1 because C4 put a higher local-pref for routes advertised by C3. Then, as the reachability check is performed, the number of routes passing through C1 increases. Eventually, all the routes that have been marked for path reachability pass through C1.

We observe that the routes are verified linearly. Since our prototype validates one path at a time and thus only `ping` one route as it goes, reachability check takes time. Nevertheless, it is possible to improve the speed of path reachability by pinging faster. Tools such as ZMap [DWH13] have shown that it is possible to `ping` the entire Internet very quickly.

Even with a faster option to validate paths, the validator will still be limited by the time to contact a destination in the dataplane. Figure 6.6 shows the Round-Trip Time (RTT) of 7k destinations scattered across the Internet. The reachable destinations were retrieved from the CAIDA dataset [CAI22]. This figure reflects the time BGP would take to validate a route with a validator over the Internet. If the destination to contact is close, the reachability check

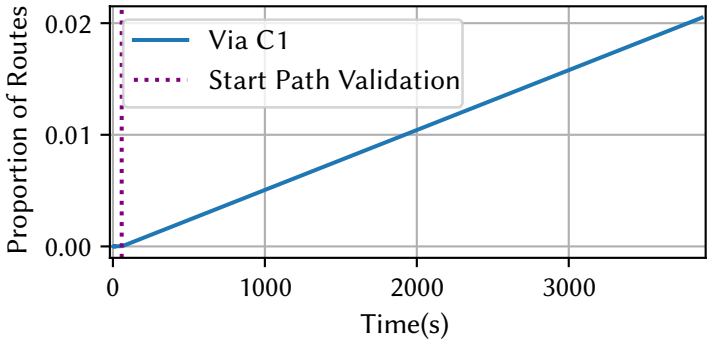


Figure 6.5: Proportion of the destinations passing via C1.

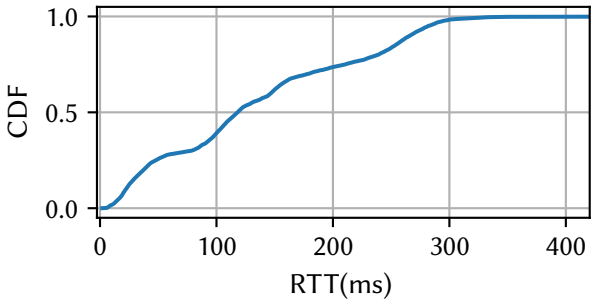


Figure 6.6: RTT of 7k IPv4 destinations evenly distributed over the Internet.

will take less time. On the contrary, if the destination is far from the validator, it will take more time to validate the prefix. However, it should be noted that more than 50% of the destinations in the Internet have an RTT of more than 120 ms, which is long to validate a BGP route. From these numbers, it seems clear that the time to validate a BGP route is increasing. However, this is the price to pay to be sure that the prefix is reachable in the dataplane.

As paths are verified in the dataplane, routing traffic increases. Our early experiments show that establishing a full TLS connection to transmit a 56-byte payload takes about 3.2 KB of traffic. To make a comparison, we computed the average BGP Update length that carries one IPv4 prefix. The BGP Update message is generated from the same RIPE RIS snapshot used in this work. In average, each BGP Update takes 185 bytes on the wire. There is clearly a significant increase in traffic generated by the router interface performing the path reachability. This will have an impact at cold boot when routers need to learn all routes from their neighbors. However, after the initial convergence, updates still occur, but at a slower rate. Recent reports [Geo22] show that routers receive an average of 0.5 prefixes update per second. This additional traffic due to path reachability check is therefore negligible compared to the

number of daily prefix updates.

6.4 Discussion

Throughout this Chapter, we propose to augment the routing decision with information from the dataplane. We argue that the linking of both control plane and dataplane can lead to a more accurate and finer-grained routing. In this section, we first discuss the implication of using active probing to check the reachability of a route. Then, we discuss the potential use of the data collected from the dataplane to implement new routing decisions.

Beacons and prefix tests Some prefixes on the Internet are advertised for analysis and testing purposes only [Mao+03]. Performing path reachability on these routes is therefore meaningless since no services are expected to operate in these special prefixes. The path reachability solution should therefore only be executed on paths for which it is explicitly asked to validate the path. That is, when sending a reachability message makes sense. To implement this idea, an RPKI object that binds the prefix to a TLS server must be explicitly created. This object indicates that the reachability check must be performed.

Detecting zombie routes and censorship A zombie route occurs when a route is still considered reachable by some routers but removed by others. This situation can occur due to a software bug where the BGP implementation fails to process a BGP Withdraw [Ong+21]. This situation leads to network degradation because part of the network will continue to use an unreachable path, which can cause traffic blackholing. Our path reachability method could periodically scan all destinations in the routing table to ensure that their routes are still reachable. If the validator detects that a route is no longer reachable, it will trigger an update by notifying its neighbors. Similarly, some ASes practice censorship on the Internet. A famous example occurred in 2008, when Pakistan Telecom announced the DNS YouTube prefix throughout the world [Gol14] to censor the platform in the country. Recently, other countries used BGP to censor part of the Internet [Sal+21]. Since this form of censorship is not detected in the control plane, adding a path validator in BGP allows routers to check if the prefix is reachable and thus prevents routers from propagating hijacked prefixes to the world.

Reconcile ASes that want to check their respective prefixes at the same time Let's imagine two Stub networks, AS1 and AS2. These two ASes use path reachability to ensure that there is a valid path between them. If both ASes receive each other's announcement at the same time, it will not be possible

to perform path reachability, since the routing table of neither ASes have an entry for the other AS. Then they have no way to return confirmation that the path is valid. When a peering link between a stub and its provider is formed, the provider will usually use an address belonging to the stub's prefix. Since the stub prefix is a subset of the provider's addressing space, the address chosen by the provider's router is reachable from both AS1 and AS2. Using the address of the provider's router, AS1 and AS2 can use the path reachability again. Back to Figure 6.1, to perform path reachability, AS1 and AS2 will need to use the addresses associated to the green interfaces of their edge routers.

Application in transit networks Doing path reachability for transit networks has to be done with great care. If a transit AS considers the prefix not reachable because of a TLS certificate incorrect configuration, no traffic will transit through the AS for the prefix. This is an important decision for a large transit AS to make. Currently, with origin validation (e.g., with RPKI), ASes have the choice to decide whether or not to reject a prefix that is invalidated by the RPKI validator. Path reachability can replicate the same behavior as implemented for route origin validation. If the reachability check detects that a path to a prefix is invalid, it should not necessarily be rejected. The operator should have the option of inserting the path in the routing table in the hope that another valid path will replace it as soon as possible.

Using other secure transport layer Throughout this chapter, we propose to use TLS to validate BGP paths. To perform the verification, the router exchanges only a few packets to check if the *Reachability Server* can respond to the query. However, TLS requires a reliable connection to be used. Maintaining a state to send only a few TLS data requires many resources that routers may not have. Instead, other secure protocols such as DTLS [RM12] or QUIC [IT21] can be used. These protocols are designed to secure data sent over an unreliable datagram connection and are therefore a good alternative to a full TLS stack for our use case. Since our prototype is quite flexible, changing the security layer is not a problem since we do not depend on any specific features of TLS. We use TLS only as a way to secure the data and authenticate the remote device.

Path Validation deployment To deploy it, there are two steps to consider. The first is to support path reachability in BGP through the new RPKI object described in Section 6.2. The second is to set up the *RS* (or its equivalent on the router) that allows reachability requests to be answered. These steps can be performed in any order and incrementally without disrupting the path reachability check operation.

In addition, similar to the current ROA deployment, all ASes can choose to enable Path Reachability check and integrate with other participating ASes. The operator can decide whether or not to accept routes from ASes that have not yet deployed an RPKI object related to path reachability.

6.5 Related Work

`LIFEGUARD` [Kat+12] actively monitors the network with `traceroutes` to PlanetLab hosts [Chu+03] to catch failures. It is able to detect the exact location of the failure in the network. When noticed, `LIFEGUARD` sends a BGP advertisement with a modified AS-Path to invalidate the failed path and force all routers in the network to select another reachable path in the dataplane. When a failure is detected in AS1, BGP `LIFEGUARD` routers re-announce their prefixes by appending AS1 in their AS-Path. When AS1 receives the poisoned AS-Path, the BGP loop detection algorithm discards the path and thus withdraws the route, forcing all downstream routers to change their routes. With our BGP Path Reachability check, the routers reconfigure themselves without poisoning the AS-Path information. This allows to be compliant with recent RFC drafts that try to verify the AS adjacencies [SsA20] or the complete AS-Path [LS17].

BGPsec [HB11], standardized by the IETF, verifies the complete AS-Path. Before announcing a BGP Update, all routers sign their messages. The AS-Path attribute is replaced by a new, more secure BGP attribute called "BGPsec Path". Upon receiving the announce, any BGP router is guaranteed that the dataplane will follow the control plane. While it verifies the integrity of the AS-Path, it cannot prevent a network failure inside an AS. Furthermore, deploying BGPsec is non-trivial and requires that all ASes participate in its deployment [LGS13]. Other studies have shown that it is still possible to create forwarding loops or perform wormhole attacks with BGPsec enabled [LHZ14]. However, BGP Path Reachabilty can be used in addition to BGPsec to make sure that the dataplane will be able to forward packets on the path announced by BGP.

BGP Path-End Validation [Coh+16] is an alternative to BGPsec. It has shown that by only ensuring that the last AS hop is valid, a comparable level of security as BGPsec can be achieved even when BGP Path-End is not fully deployed. There is no need to replace routers to deploy BGP Path-End since the router does not need to compute cryptographic signatures in BGP messages. They only need a new RPKI object to work. Just like BGPsec, BGP Path-End validation only helps the control plane integrity but does not check the current state of the dataplane. These verification techniques do not guarantee that the path taken by the traffic will not result in losses or failures.

Instead of solving the security problems of BGP, `SCION` [Zha+11] proposes a new Internet architecture to overcome the current limitations of BGP. `SCION`

uses a cryptographic authentication system to avoid configuration errors, route leaks or prefix hijacking, thus ensuring connectivity in the dataplane.

Other tools to scan the Internet have been developed to examine the general state of the network. Thunderping [Pad+19] or Trinocular [QHP13] detect local and global problems respectively. Another tool uses RIPE Atlas [RIP23] data to monitor the dataplane health [Fon+17]. By actively scanning the reachable destinations, these tools are able to accurately detect problems in the network. To improve the implementation of BGP path reachability, the techniques developed in these tools could be used.

6.6 Conclusion

We introduced BGP Path Reachability, a new verification system to check if a route is reachable in the dataplane. Before adding a route to the routing table, it will first contact a specialized service responsible for responding to the router's request. With this method, routing errors due to misconfiguration can be reduced because the router is now sure that the traffic can reach the destination AS and will not be discarded on the way. We have demonstrated the feasibility of this approach by proposing a working prototype implemented in the FRRouting suite. Finally, we discussed the possibility of bringing dataplane information into the routing world. This would provide a more accurate view of the network and therefore allow a more relevant and optimal path to be chosen. Similarly, the control plane can be notified when an event occurs in the dataplane to adapt the routing decision.

Part V

Future Directions and Conclusion

Discussion & Future Directions

7

Throughout this thesis, several potential research directions were opened up, which could serve as a basis for future research. This chapter briefly introduces future potential directions for the concepts discussed within this thesis.

Protocol Extensibility

In Part II of this thesis, we present *x*BGP, a new paradigm that enables network operators to program their routers. With *x*BGP, operators can create extensions that run on any routing protocol implementation. This opens up many possibilities for improving and extending protocols in the future.

The design of future routing protocols. A first potential area for future research is to explore how future protocols can take advantage of this scalability and programmability. Most standardized protocols are designed by committees. This process has advantages and drawbacks. By involving more people in their design, the new protocols have a higher probability of meeting the user/operator's requirements. However, these committees sometimes tend to over specify and add new features that require long discussions before reaching an agreement. The design of an extensible routing protocol could be done differently. A small committee could start with a small set of basic functionalities, a virtual machine and a simple API. Based on these specifications, the community could then openly develop extensions to meet their specific requirements. Instead of trying to match all the expressed requirements, the protocol designers would focus on getting the basic principles of the core protocol right. If some of these extensions become popular or require better performance than the one achievable with virtual machines, they could later be included in a new version of the protocol.

Designing a protocol fully compatible with *x*BGP. A second area of future research focuses on the design of BGP implementations that better support *x*BGP. Currently, *x*BGP interfaces with BGP implementations such as BIRD and FRRouting. However, modifications are required to make these implementations compatible with *x*BGP, such as modifying internal function

signatures, adding missing data to structures or including additional arguments in functions. In addition, implementations must incorporate an overlay to enable *x*BGP vendor-neutral data. This results in additional memory usage and slower execution of *x*BGP programs.

One possible solution is to redesign the underlying protocol implementation to improve the efficiency of *x*BGP integration. This involves designing the implementation so that it requires minimal code modifications to support *x*BGP and minimizes the transformation of structures passed to *x*BGP programs.

Using other virtual machines. A third research direction is related to the virtual machine used. eBPF was the most mature virtual machine during the development of *x*BGP. However, other virtual machines such as WebAssembly seem more promising and start to perform well. It might be interesting to see the advantages of using them in the context of *x*BGP.

WebAssembly [Haa+17] does not have the same origins as eBPF, since it was designed to program "high-performance" applications for the web. Although WebAssembly is not limited to the web, it is designed to achieve high performance similar to native code, to guarantee security through validated code execution in a secure, isolated environment, and to ensure portability across hardware, languages and platforms.

Unlike eBPF, WebAssembly operates in a 32-bit address space. It has a dedicated linear memory that cannot be escaped, in addition to a local stack. A WebAssembly program also includes functions, tables and global variables that can be accessed by other WebAssembly programs.

WebAssembly seems to be well suited to the requirements of an *x*BGP program due to its memory isolation and programming interface characteristics. In addition, the recent introduction of WASIX [Was23a], which makes it possible to create fully POSIX-compliant applications, e.g., by implementing part of the `libc` [Was23b], offers increased functionality for the development of more sophisticated *x*BGP programs.

Using other programing language for *x*BGP programs In Chapter 4, we explained that *x*BGP programs are written in C, which is considered as an unsafe language. To address this concern, we used several software verification tools to check unsafe aspects of the language. However, Rust, a relatively new programming language, is gaining in popularity in recent years [Cor23]. When a Rust program is compiled, it enforces that all references (unless explicitly marked as unsafe) point to valid memory locations. By combining Rust's memory safety guarantee with WebAssembly's features, it could become possible to achieve robust memory guarantees for programs running in a routing protocol implementation.

Leveraging Secure Transport

In Part III, we have demonstrated three new features that improve BGP. We believe that using QUIC as the transport layer of routing protocols can bring additional benefits to distributed routing protocols. Our current prototypes are a first step to reach this goal. They open new directions that researchers, network operators and eventually the IETF will need to explore. We discuss some of these directions in this section.

A precise semantics for the BGP over QUIC certificates. Our BGP over QUIC (BoQ) prototype uses some fields of the X.509 certificates to distribute prefix lists, filters and other types of information. Network operators and the IETF should discuss and agree on a semantics that defines the information which can be exchanged inside the BGP over QUIC certificates. We showed in Section 5.5.1 how X.509 certificates can be used to authenticate additional routing information, especially in the case of open-peering policies. One could extend such use case by embedding additional information. These certificates could carry different types of information, from prefix lists to traffic engineering requirements. For example, a field in the certificate could indicate the type of peering (shared cost, client, open-peering, ...) and the router receiving such authenticated information could dynamically accept or deny the connection request. Furthermore, edge routers of an AS could automatically modify the configuration of their import and export filters according to the requesting router type (stub, transit, ...).

Certification authorities and revocation. Using X.509 certificates to support BGP will bring new operational concerns as for the current public key infrastructure [For+03] and its certificate authorities. BGP over QUIC would benefit from a public PKI, which could be built upon the RPKI [MK20], to certify that a router belongs to a given AS. This PKI could be provided by a coalition of an Internet Routing Registries (IRR) such as APNIC, AFNIC, RIPE, ...

Hence, BoQ could be directly integrated with existing RPKIs without major infrastructure changes. Edge routers could dynamically verify, upon connection initiation, that their peer effectively belongs to the expected AS.

When network operators start to use certificates, they also need to define procedures to manage the expiration of the revocation of these certificates. The procedures used for the web PKI might not all be applicable to certificates used by routing protocols.

Another existing piece of PKI that could be reused as-is is the Online Certificate Status Protocol (OCSP) [San+13]. This allows the management of certificates lifetimes with Certificates Revocation Lists (CRLs). Routers could

directly verify, upon connection initiation, if the remote peer still has a valid certificate. By revoking a router's certificate, network operator could remove routers or a whole peering region.

Path diversity strategies with *x*BGP. *x*BGP (Chapter 4) adds flexibility to BGP by offering a common interface letting operators implement their own BGP extensions. Such tool could be used to implement advanced path selection policies as discussed in Section 5.5.2. However, this would require porting *x*BGP over our BoQ prototype to benefit from QUIC since the current version is built on top of a plain implementation of BGP. This problem can be explored in a future BoQ prototype. Currently, if BoQ plans to use certificates for a specific use case, it is mandatory that the semantics are understood by the BGP implementation that receives the certificate. Consequently, the code responsible for managing the directives described in the certificate must be developed in the BGP implementation itself. This would be difficult to manage as path selection policies may vary according to operators. With the aid of *x*BGP, it is now conceivable that an X.509 certificate ships *x*BGP programs in addition to the predefined directives listed in the certificate.

After receiving the certificate containing *x*BGP programs, a *x*BGP-compatible router could inject them to be executed during the BGP session connecting the two BGP speakers. This would grant significant flexibility in designing use-cases that could be implemented using xBGP on top of BoQ, thus enabling custom path selection by a remote AS.

Making routing protocol "data-plane" aware

The final contribution of this thesis (Part IV) focused on the verification of BGP routes through the establishment of a secure handshake in the data-plane. This approach is in line with the objective of improving the dynamicity of the control-plane as a function of events occurring in the data-plane. In this section, we examine some of these directions and considerations.

Augmenting the BGP Decision Process. The BGP decision process traditionally relies on control plane information to select the best route to a destination. However, this approach often results in sub-optimal decisions. Integrating data plane metrics, such as latency, into the decision process can lead to more optimal path choices. Studies have shown that taking latency into account can improve the decision for around 77% of prefixes [Nak+22].

Network providers often use virtual routing and forwarding tables (VRFs) to segment a physical router into several virtual routers. This approach is commonly adopted by operators to manage various customer services, particularly in cases where each customer has several remote sites linked only

by their network provider. By using VRFs, operators can maintain a logical separation between different customers, allowing each customer to have its own isolated routing environment. This enables efficient management and control of customer-specific routing policies, guaranteeing secure and reliable connectivity between the customer's remote sites and their associated hosts.

By developing this concept further, we can consider integrating other data-plane metrics, such as bandwidth throughput, to form multiple VRFs per metric. For example, if an operator would like to maximize bandwidth for specific prefixes, they can configure their BGP routers accordingly to reflect this policy. Similarly, another operator can configure other prefixes associated with video traffic to minimize latency. By doing so, the operator has the ability to apply for a variety of routing policies that can be adjusted to optimize network performance according to their specific needs.

With several optimized virtual routing tables at their disposal, the network operator can selectively choose which routes to advertise and use for packet transmission. In addition, a network can offer a service to its BGP peers by enabling them to advertise their prefixes with a specific metric optimization objective. This can be facilitated by using an arbitrary BGP community. For example, a customer can send a BGP route with an arbitrary BGP community to the data plane-aware provider, asking it to advertise the prefix on low latency paths.

This introduces a new type of "control-plane-oriented Quality of Service (QoS)", which complements existing QoS techniques implemented in the data plane. The combination of control plane and data plane QoS enables a more comprehensive and customizable approach to network optimization.

Enabling fine-grained Traffic Engineering methods. BGP typically selects for each destination a single next hop, or several ones in the case of ECMP (Equal-Cost Multi-Path) routing. However, the same IP prefix can provide access to different targets, such as services with specific requirements, e.g., low latency or low loss rate. With BGP alone, network operators cannot force one exit path over another without resorting to complex traffic engineering techniques. Depending on the exit AS or exit router, the choice of specific paths may allow optimizing specific metrics, e.g., bandwidth, latency, etc.

BGP-Egress Peer Engineering (BGP-EPE) [Fil+21b] is a technique that relies on Segment Routing over IPv6 (SRv6) [Fil+21a] to force the forwarding path from a AS border router. When a BGP-EPE enabled router receives a packet encapsulated in an SRv6 header, it can discriminate the path that packets should take based on the Segment Identifier (SID).

By combining an approach that takes into account data-plane metrics with BGP-EPE, network operators can forward data-plane traffic according to the metric they wish to optimize. This integration enables more granular control

on routing decisions.

Using data-plane data to influence the control-plane. In our previous discussion, we explored how information from the data-plane can inform BGP route selection. Conversely, we can also use the reverse approach, where events occurring in the data plane trigger notifications to the control plane for problem resolution. For example, if we find that the latency of the currently selected route exceeds a certain threshold, we can initiate a new route selection by invoking the BGP decision process. This proactive approach enables the control plane to dynamically adapt to changing data-plane conditions and make more informed routing decisions to optimize network performance.

Conclusion 8

Distributed routing protocols, which constitute a large fraction of today's Internet control-plane, no longer fully meet the requirements of modern networks. These protocols originated in the early days of the Internet, in the late 1980s, and despite some extensions through RFC standardization over time, they have become rigid and resistant to change. This lack of flexibility makes it difficult to propose improvements or modify implementations. A notable example of this ossification occurred in 2010 when several BGP routers on the Internet crashed following a routing experiment conducted by a research team at Duke University [RIP10]. The purpose of the experiment was to evaluate the support for 32-bit AS numbers throughout the Internet.

This thesis aimed to improve distributed routing protocols by addressing three key aspects that correspond to the current requirements of modern networks.

In Part II, we focused on the programmability of routing protocols. Instead of depending solely on router vendors, operators can dynamically modify their routing protocols without having to reboot or recompile routers. To this end, we have integrated an eBPF-based virtual machine into the BGP protocol implementation, enabling the execution of arbitrary code written by a network operator. We demonstrated the feasibility of this approach by implementing various use cases using the eBPF virtual machine we integrated in BGP and OSPF. Chapter 4 took the notion of programmable protocols a step further by proposing the concept of *x*BGP, which enables programmability within the BGP protocol itself, independently of the underlying implementation. *x*BGP consists of (*i*) a virtual machine, (*ii*) an API to communicate with the host BGP implementation, and (*iii*) dedicated insertion points where *x*BGP programs can be executed. With *x*BGP, a single plugin can be written once can run on any *x*BGP-compatible BGP implementation. We have validated this new approach by making two open-source BGP implementations, BIRD and FRRouting, compatible with *x*BGP and by implementing use cases that are difficult to reproduce on traditional BGP implementations.

Then, In the Part III, we focused on modernizing the transport of routing messages. The BGP protocol currently relies on a raw TCP session to exchange messages. However, we have shown that TCP presents several significant problems when used in the context of the BGP protocol. The main problem is

its lack of security, as it makes the BGP session vulnerable to potential attacks that can lead to traffic disruptions, such as blackholing.

To solve this problem, we proposed replacing TCP by QUIC, a UDP-based transport protocol. QUIC includes encryption and other features designed to enhance those already integrated inside TCP. By using a secure transport protocol like QUIC into BGP, we noticed a potential in the way QUIC authenticates routers. Indeed, one notable advantage is that QUIC uses TLS 1.3, which relies on X.509 certificates for authentication between parties involved in the QUIC connection. By relying on these certificates, we have demonstrated the ability (*i*) to automatically reconfigure routers based on the information provided by the certificates, (*ii*) to offer a more secure blackholing service and (*iii*) to establish BGP sessions on demand according to load and network status.

By replacing the transport layer of routing protocols with QUIC, we believe that our results may encourage the IETF to move the ongoing BGP over QUIC draft towards its standardization. This would allow researcher and protocol designers to explore new routing approaches.

Finally, in Part IV we focused on a specific aspect of BGP route security. When BGP announces routes, there is no guarantee that they can be reached and used. This is because BGP relies solely on control-plane information to determine the best route, without taking into account the current state of the network. In Chapter 6, we developed a method for checking whether a BGP route is reachable in the data-plane before adding it to the router's routing table and announcing it to other BGP speakers. We tested our approach in FRRouting and carried out a simple experiment which showed that BGP no longer selects unreachable routes in the data plane, thus ensuring connectivity for all advertised prefixes. Naturally, this method slows down the routing decision process, as our prototype needs to establish a secure exchange to validate the routing. However, this delay is necessary to guarantee route reachability.

This new approach represents a first step towards leveraging the potential of integrating data-plane metrics into the control-plane. It demonstrates how integrating these metrics can enhance the flexibility of the control-plane. In Chapter 7, we presented various directions for future research and highlighted unresolved questions in this area. The objective was to encourage other researchers to build on the foundations laid by this work and delve deeper into this subject.

Bibliography

[ABD14] L. Alt, R. Beverly, and A. Dainotti. "Uncovering network tarpits with degreaser". In: *Proceedings of the 30th Annual Computer Security Applications Conference*. 2014, pp. 156–165.

[ACT13] P. Aitken, B. Claise, and B. Trammell. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. RFC 7011. Sept. 2013. DOI: 10.17487/RFC7011.

[Ael20] M. Aelmans. *The TCP Authentication Option (TCP-AO)*. 2020. URL: https://conference.apnic.net/50/assets/files/APCS790/The-TCP-Authentication-Option.pdf (visited on 06/22/2023).

[And+10] L. Andrew et al. "Flexible BGP Communities". Internet draft, draft-lange-flexible-bgp-communities-03, work in progress. Aug. 2010.

[AP15] G. Aceto and A. Pescapé. "Internet Censorship detection: A survey". In: *Computer Networks* 83 (2015), pp. 381–421. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2015.03.008.

[ARS18] R. Amin, M. Reisslein, and N. Shah. "Hybrid SDN Networks: A Survey of Existing Approaches". In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 3259–3306. DOI: 10.1109/COMST.2018.2837161.

[Azi+20] A. Azimov, E. Bogomazov, R. Bush, K. Patel, and J. Snijders. *Verification of AS_PATH Using the Resource Certificate Public Key Infrastructure and Autonomous System Provider Authorization*. Internet-Draft draft-ietf-sidrops-aspa-verification-04. Work in Progress. Internet Engineering Task Force, Mar. 2020. 12 pp.

[Azi+22] A. Azimov, E. Uskov, R. Bush, K. Patel, J. Snijders, and R. Housley. *A Profile for Autonomous System Provider Authorization*. Internet-Draft draft-ietf-sidrops-aspa-profile-07. Work in Progress. Internet Engineering Task Force, Jan. 2022. 9 pp.

[Azi+23a] A. Azimov, E. Bogomazov, R. Bush, K. Patel, J. Snijders, and K. Sriram. *BGP AS_PATH Verification Based on Autonomous System Provider Authorization (ASPA) Objects*. Internet-Draft draft-ietf-sidrops-aspa-verification-16. Work in Progress. Internet Engineering Task Force, Aug. 2023. 23 pp.

[Azi+23b] A. Azimov, E. Uskov, R. Bush, J. Snijders, R. Housley, and B. Maddison. *A Profile for Autonomous System Provider Authorization*. Internet-Draft draft-ietf-sidrops-aspa-profile-16. Work in Progress. Internet Engineering Task Force, July 2023. 14 pp.

[AZV17] M. Apostolaki, A. Zohar, and L. Vanbever. "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 375–392. DOI: 10.1109/SP.2017.29.

[BA13] R. Bush and R. Austein. *The Resource Public Key Infrastructure (RPKI) to Router Protocol*. RFC 6810. Jan. 2013. DOI: 10.17487/RFC6810.

[BA17] R. Bush and R. Austein. *The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1*. RFC 8210. Sept. 2017. DOI: 10.17487/RFC8210.

[BDL10] R. Blair, A. Durai, and J. Lautmann. *Tcl scripting for Cisco IOS*. Cisco Press, 2010.

[Bec+17a] R. Beckett, A. Gupta, R. Mahajan, and D. Walker. "A general approach to network configuration verification". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 155–168.

[Bec+17b] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker. "Network configuration synthesis with abstract topologies". In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2017, pp. 437–451.

[Bel58] R. Bellman. "On a Routing Problem". In: *Quarterly of Applied Mathematics* 16.1 (Apr. 1958). Full publication date: APRIL, 1958, pp. 87–90.

[Ber+18a] M. Bertrone, S. Miano, F. Risso, and M. Tumolo. "Accelerating Linux Security with EBPF Iptables". In: *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. SIGCOMM '18. Budapest, Hungary: Association for Computing Machinery, 2018, pp. 108–110. ISBN: 9781450359153. DOI: 10.1145/3234200.3234228.

[Ber+18b] M. Bertrone, S. Miano, F. Risso, and M. Tumolo. "Accelerating Linux Security with eBPF iptables". In: *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. ACM. 2018, pp. 108–110.

[BH22] R. Bush and R. Housley. *The 'I' in RPKI Does Not Stand for Identity*. RFC 9255. June 2022. DOI: 10.17487/RFC9255.

[Bir+19] H. Birge-Lee, L. Wang, J. Rexford, and P. Mittal. "Sico: Surgical interception attacks by manipulating BGP communities". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 431–448.

[Boe+08] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. May 2008. DOI: 10.17487/RFC5280.

[Bon+22] M. Bonola, G. Belocchi, A. Tulumello, M. S. Brunella, G. Siracusano, G. Bianchi, and R. Bifulco. "Faster Software Packet Processing on FPGA NICs with eBPF Program Warping". In: *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, July 2022, pp. 987–1004. ISBN: 978-1-939133-29-58.

[Bos+14] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 87–95.

[Bot+14] F. Botelho, A. Bessani, F. M. Ramos, and P. Ferreira. "On the Design of Practical Fault-Tolerant SDN Controllers". In: *2014 Third European Workshop on Software Defined Networks*. 2014, pp. 73–78. DOI: 10.1109/EWSDN.2014.25.

[BPT15] M. Belshe, R. Peon, and M. Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. May 2015. DOI: 10.17487/RFC7540.

[BPW19] R. Bush, K. Patel, and D. Ward. *Extended Message Support for BGP*. RFC 8654. Oct. 2019. DOI: 10.17487/RFC8654.

[Bre+20] J. Brenes, A. García-Martínez, M. Bagnulo, A. Lutu, and C. Pelsser. "Power Prefixes Prioritization for Smarter BGP Reconvergence". In: *IEEE/ACM Transactions on Networking* 28.3 (2020), pp. 1074–1087. DOI: 10.1109/TNET.2020.2979665.

[Bro+16] M. Brockschmidt, B. Cook, S. Ishtiaq, H. Khlaaf, and N. Piterman. "T2: temporal property verification". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2016, pp. 387–393.

[Bru+17] T. Bruijnzeels, O. Muravskiy, B. Weber, and R. Austein. *The RPKI Repository Delta Protocol (RRDP)*. RFC 8182. July 2017. DOI: 10.17487/RFC8182.

[BSM18] F. Bannour, S. Souihi, and A. Mellouk. "Distributed SDN Control: Survey, Taxonomy, and Challenges". In: *IEEE Communications Surveys & Tutorials* 20.1 (2018), pp. 333–354. DOI: 10.1109/COMST.2017.2782482.

[Cae+05] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. "Design and implementation of a routing control platform". In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association. 2005, pp. 15–28.

[CAI22] CAIDA. *The CAIDA Macroscopic Internet Topology Data Kit - 2021-03*. 2022. URL: https://www.caida.org/catalog/datasets/internet-topology-data-kit (visited on 05/24/2022).

[Cal06] K. Calvert. "Reflections on network architecture: an active networking perspective". In: *ACM SIGCOMM Computer Communication Review* 36.2 (2006), pp. 27–30.

[Cas+14] I. Castro, J. C. Cardona, S. Gorinsky, and P. Francois. "Remote peering: More peering without internet flattening". In: *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 2014, pp. 185–198.

[Cas+20] C. Cassagnes, L. Trestioreanu, C. Joly, and R. State. "The rise of eBPF for non-intrusive performance monitoring". In: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. 2020, pp. 1–7. DOI: 10.1109/NOMS47738.2020.9110434.

[CBC06] E. Chen, T. J. Bates, and R. Chandra. *BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)*. RFC 4456. July 2006. DOI: 10.17487/RFC4456.

[CCI84] CCITT. *Message Handling Systems: Presentation Transfer Syntax and Notation*. Recommendations X.409. Oct. 1984.

[CCI88] CCITT. "Recommendation X.208, Specification of Abstract Syntax Notation One (ASN.1)". In: *Data Communications Networks Open Systems Interconnection (OSI) Model and Notation, Service Definition, Blue Book. Vol VIII* (Nov. 1988).

[CCI89] CCITT. "Recommendation X.509, The Directory-Authentication Framework, Blue-Book-Melbourne (1988), Fascicle VIII. 8: Data Communication Networks: Directory". In: *International Telecommunications Union, Geneva, Switzerland* (1989), pp. 127–141.

[CCL19] B. Claise, J. Clarke, and J. Lindblad. *Network Programmability with YANG*. Addison-Wesley, 2019.

[Che00] E. Chen. *Route Refresh Capability for BGP-4*. RFC 2918. Sept. 2000. DOI: 10.17487/RFC2918.

[Cho+19] S. Cho, R. Fontugne, K. Cho, A. Dainotti, and P. Gill. "BGP hijacking classification". In: *2019 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE. 2019, pp. 25–32. DOI: 10.23919/TMA.2019.8784511.

[Chu+03] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. "PlanetLab: An Overlay Testbed for Broad-Coverage Services". In: *SIGCOMM Comput. Commun. Rev.* 33.3 (July 2003), pp. 3–12. ISSN: 0146-4833. DOI: 10.1145/956993.956995.

[Chu+19] T. Chung, E. Aben, T. Bruijnzeels, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, R. v. Rijswijk-Deij, J. Rula, et al. "RPKI is Coming of Age: A Longitudinal Study of RPKI Deployment and Invalid Route Origins". In: *Proceedings of the Internet Measurement Conference*. 2019, pp. 406–419.

[Cis05] Cisco Systems, Inc. *Remotely Triggered Black Hole Filtering – Destination Based and Source Based*. White paper. 2005.

[Cis11] Cisco. *Cisco IOS Scripting with TCL Configuration Guide*. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ios_tcl/configuration/12-4t/ios-tcl-12-4t-book/nm-script-tcl.html. 2011.

[Cis15] I. Cisco Systems. *Route-Maps for IP Routing Protocol Redistribution Configuration*. 2015. URL: https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/49111-route-map-bestp.html (visited on 05/20/2022).

[Cis20] Cisco. *Cisco Annual Internet Report (2018–2023) White Paper*. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. 2020.

[Cit+10] L. Cittadini, W. Muhlbauer, S. Uhlig, R. Bush, P. Francois, and O. Maennel. "Evolution of internet address space deaggregation: myths and reality". In: *IEEE Journal on Selected Areas in Communications* 28.8 (2010), pp. 1238–1249.

[Clo] Cloudflare. "Quiche". `https://github.com/cloudflare/quiche`.

[Coh+16] A. Cohen, Y. Gilad, A. Herzberg, and M. Schapira. "Jumpstarting BGP security with path-end validation". In: *Proceedings of the 2016 ACM SIGCOMM Conference*. SIGCOMM '16. Florianopolis, Brazil: Association for Computing Machinery, 2016, pp. 342–355. ISBN: 9781450341936. DOI: 10.1145/2934872.2934883.

[Cor23] J. Corbet. "A first look at Rust in the 6.1 kernel". `https://lwn.net/Articles/910762/`. Oct. 2023.

[CPR06] B. Cook, A. Podelski, and A. Rybalchenko. "Terminator: beyond safety". In: *International Conference on Computer Aided Verification*. Springer. 2006, pp. 415–418.

[CS21] J. Chroboczek and D. Schinazi. *The Babel Routing Protocol*. RFC 8966. Jan. 2021. DOI: 10.17487/RFC8966.

[CSR16] E. Chen, N. Shen, and R. Raszuk. *Carrying Geo Coordinates in BGP*. Internet-Draft draft-chen-idr-geo-coordinates-02. Work in Progress. Internet Engineering Task Force, Oct. 2016. 9 pp.

[Cur+23] L. Curley, K. Pugin, S. Nandakumar, and V. Vasiliev. *Warp - Live Media Transport over QUIC*. Internet-Draft draft-lcurley-warp-04. Work in Progress. Internet Engineering Task Force, Mar. 2023. 30 pp.

[CZN20] CZ.NIC, z.s.p.o. "BIRD Internet Routing Daemon". `https://gitlab.nic.cz/labs/bird`. 2020. (Visited on 05/25/2022).

[Dav04] G. Davies. *Designing and Developing Scalable IP Networks*. John Wiley & Sons, 2004.

[DB08] B. Donnet and O. Bonaventure. "On BGP communities". In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 55–59.

[De +19] Q. De Coninck, F. Michel, M. Piraux, F. Rochet, T. Given-Wilson, A. Legay, O. Pereira, and O. Bonaventure. "Pluginizing QUIC". In: *Proceedings of the ACM Special Interest Group on Data Communication*. 2019, pp. 59–74.

[Dec+23] B. Decraene, L. Ginsberg, T. Li, G. Solignac, M. Karasek, C. Bowers, G. V. de Velde, P. Psenak, and T. Przygienda. *IS-IS Fast Flooding*. Internet-Draft draft-ietf-lsr-isis-fast-flooding-03. Work in Progress. Internet Engineering Task Force, Mar. 2023. 25 pp.

[Dec+98] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner. "Router Plugins: A Software Architecture for next Generation Routers". In: *SIGCOMM Comput. Commun. Rev.* 28.4 (Oct. 1998), pp. 229–240. ISSN: 0146-4833. DOI: 10.1145/285243.285285.

[Dev20] C. DevNet. *Application Hosting*. 2020. URL: https://developer.cisco.com/docs/ios-xe/%5C#!application-hosting-quick-start-guide/application-hosting-options (visited on 05/24/2022).

[DH76] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[Dij59] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1.1 (Dec. 1959), pp. 269–271. ISSN: 0945-3245. DOI: 10.1007/BF01386390.

[Don17] J. A. Donenfeld. "Wireguard: next generation kernel network tunnel." In: *NDSS*. 2017, pp. 1–12.

[DSM19] L. Deri, S. Sabella, and S. Mainardi. "Combining System Visibility and Security Using eBPF". In: *Proceedings of the Third Italian Conference on Cyber Security (ITASEC)*. ITASEC '19. 2019, pp. 50–62.

[DWH13] Z. Durumeric, E. Wustrow, and J. A. Halderman. "ZMap: Fast Internet-Wide Scanning and Its Security Applications". In: *Proceedings of the 22nd USENIX Conference on Security*. SEC'13. Washington, D.C.: USENIX Association, 2013, pp. 605–620. ISBN: 978-1-931971-03-4.

[EF10] J. W. Evans and C. Filsfils. *Deploying IP and MPLS QoS for multiservice networks: Theory and practice*. Elsevier, 2010.

[Enn+11] R. Enns, M. Björklund, A. Bierman, and J. Schönwälder. *Network Configuration Protocol (NETCONF)*. RFC 6241. June 2011. DOI: 10.17487/RFC6241.

[ER13] J. Erman and K. K. Ramakrishnan. "Understanding the super-sized traffic of the super bowl". In: *Proceedings of the 2013 conference on Internet measurement conference*. 2013, pp. 353–360.

[Fan+09] M. Fanto, R. Atkinson, M. Barnes, V. Manral, R. White, T. Li, and M. Bhatia. *OSPFv2 HMAC-SHA Cryptographic Authentication*. RFC 5709. Oct. 2009. DOI: 10.17487/RFC5709.

[FB05] N. Feamster and H. Balakrishnan. "Detecting BGP configuration faults with static analysis". In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association. 2005, pp. 43–56.

[FBR03] N. Feamster, J. Borkenhagen, and J. Rexford. "Guidelines for interdomain traffic engineering". In: *ACM SIGCOMM Computer Communication Review* 33.5 (2003), pp. 19–30.

[FC19] L. Fontana and D. Calavera. *Linux Observability with BPF*. O'Reilly, 2019.

[Fed+90] M. Fedor, M. L. Schoffstall, J. R. Davin, and D. J. D. Case. *Simple Network Management Protocol (SNMP)*. RFC 1157. May 1990. DOI: 10.17487/RFC1157.

[Fei06] M. Feilner. *OpenVPN: Building and Integrating Virtual Private Networks: Learn How to Build Secure VPNs Using This Powerful Open Source Application*. Packt Publishing, 2006. ISBN: 190481185X.

[Fil+21a] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li. *Segment Routing over IPv6 (SRv6) Network Programming*. RFC 8986. Feb. 2021. DOI: 10.17487/RFC8986.

[Fil+21b] C. Filsfils, S. Previdi, G. Dawra, E. Aries, and D. Afanasiev. *Segment Routing Centralized BGP Egress Peer Engineering*. RFC 9087. Aug. 2021. DOI: 10.17487/RFC9087.

[FK11] S. Frankel and S. Krishnan. *IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap*. RFC 6071. Feb. 2011. DOI: 10.17487/RFC6071.

[Fle17] M. Fleming. "A thorough introduction to eBPF". In: *Linux Weekly News* (Dec. 2017). https://old.lwn.net/Articles/740157/.

[FLM08] D. Ferguson, A. Lindem, and J. Moy. *OSPF for IPv6*. RFC 5340. July 2008. DOI: 10.17487/RFC5340.

[Flo02] S. Floyd. *Inappropriate TCP Resets Considered Harmful*. RFC 3360. Aug. 2002. DOI: 10.17487/RFC3360.

[Fog+15] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein. "A General Approach to Network Configuration Analysis". In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 469–483. ISBN: 978-1-931971-218.

[Fon+17] R. Fontugne, C. Pelsser, E. Aben, and R. Bush. "Pinpointing Delay and Forwarding Anomalies Using Large-Scale Traceroute Measurements". In: *Proceedings of the 2017 Internet Measurement Conference*. IMC '17. London, United Kingdom: Association for Computing Machinery, 2017, pp. 15–28. ISBN: 9781450351188. DOI: 10.1145/3131365.3131384.

[Fon+19] R. Fontugne, E. Bautista, C. Petrie, Y. Nomura, P. Abry, P. Gonçalves, K. Fukuda, and E. Aben. "BGP zombies: An analysis of beacons stuck routes". In: *International Conference on Passive and Active Network Measurement*. Ed. by D. Choffnes and M. Barcellos. Springer. Cham: Springer International Publishing, 2019, pp. 197–209. ISBN: 978-3-030-15986-3.

[fon18] foniod. "Rust library for building and running BPF/eBPF modules". https://github.com/foniod/redbpf. 2018.

[For+03] D. W. S. Ford, D. S. Chokhani, S. S. Wu, R. V. Sabett, and C. (R. Merrill. *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework*. RFC 3647. Nov. 2003. DOI: 10.17487/RFC3647.

[Fou17] T. L. Foundation. "FRRouting Project". https://frrouting.org/. 2017. (Visited on 05/25/2022).

[FR07] N. Feamster and J. Rexford. "Network-wide prediction of BGP routes". In: *IEEE/ACM Transactions on Networking (TON)* 15.2 (2007), pp. 253–266.

[Frö+17] A. Frömmgen, A. Rizk, T. Erbshäußer, M. Weller, B. Koldehofe, A. Buchmann, and R. Steinmetz. "A programming model for application-defined multipath TCP scheduling". In: *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. ACM. 2017, pp. 134–146.

[FRZ14] N. Feamster, J. Rexford, and E. Zegura. "The road to SDN: an intellectual history of programmable networks". In: *ACM SIGCOMM Computer Communication Review* 44.2 (2014), pp. 87–98.

[FSR11] A. Fabrikant, U. Syed, and J. Rexford. "There's something about MRAI: Timing diversity can exponentially worsen BGP convergence". In: *2011 Proceedings IEEE INFOCOM*. 2011, pp. 2975–2983. DOI: 10.1109/INFCOM.2011.5935139.

[Fuj+23] T. Fujita et al. "GoBGP". https://github.com/osrg/gobgp. 2023.

[Gas+18] O. Gasser, Q. Scheitle, P. Foremski, Q. Lone, M. Korczyński, S. D. Strowes, L. Hendriks, and G. Carle. "Clusters in the expanse: Understanding and unbiasing IPv6 hitlists". In: *Proceedings of the Internet Measurement Conference 2018*. 2018, pp. 364–378.

[GB19] A. García-Martínez and M. Bagnulo. "Measuring BGP Route Propagation Times". In: *IEEE Communications Letters* 23.12 (2019), pp. 2432–2436. DOI: 10.1109/LCOMM.2019.2945964.

[Geo22] Geoff Huston. *The BGP Instability Report*. 2022. URL: https://bgpupdates.potaroo.net/instability/bgpupd.html (visited on 06/29/2022).

[Ger+19] E. Gershuni et al. "Simple and Precise Static Analysis of Untrusted Linux Kernel Extensions". In: *PLDI'19*. https://research.vmware.com/publications/simple-and-precise-static-analysis-of-untrusted-linux-kernel-extensions. June 2019, pp. 1069–1084.

[Gin+19] L. Ginsberg, S. Previdi, Q. Wu, J. Tantsura, and C. Filsfils. *BGP - Link State (BGP-LS) Advertisement of IGP Traffic Engineering Performance Metric Extensions*. RFC 8571. Mar. 2019. DOI: 10.17487/RFC8571.

[Gio+17] V. Giotsas, G. Smaragdakis, C. Dietzel, P. Richter, A. Feldmann, and A. Berger. "Inferring BGP blackholing activity in the Internet". In: *Proceedings of the 2017 Internet Measurement Conference*. ACM. 2017, pp. 1–14.

[Gio+20] V. Giotsas, G. Nomikos, V. Kotronis, P. Sermpezis, P. Gigis, L. Manassakis, C. Dietzel, S. Konstantaras, and X. Dimitropoulos. "O peer, where art thou? Uncovering remote peering interconnections at IXPs". In: *IEEE/ACM Transactions on Networking* 29.1 (2020), pp. 1–16.

[Gol14] S. Goldberg. "Why is It Taking so Long to Secure Internet Routing?" In: *Commun. ACM* 57.10 (Sept. 2014), pp. 56–63. ISSN: 0001-0782. DOI: 10.1145/2659899.

[Goo+03] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. D. McDaniel, and A. D. Rubin. "Working around BGP: an incremental approach to improving security and accuracy in interdomain routing." In: *NDSS*. Vol. 23. 2003, p. 156.

[Got+03] J. Gottlieb, A. Greenberg, J. Rexford, and J. Wang. "Automated provisioning of BGP customers". In: *IEEE network* 17.6 (2003), pp. 44–55.

[Gre+05] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. "A clean slate 4D approach to network control and management". In: *ACM SIGCOMM Computer Communication Review*. CoNEXT '18 35.5 (Oct. 2005), pp. 41–54. ISSN: 0146-4833. DOI: 10.1145/1096536.1096541.

[Gre19] B. Gregg. *BPF Performance Tools*. Addison-Wesley Professional, 2019.

[GS05] T. G. Griffin and J. L. Sobrinho. "Metarouting". In: *ACM SIGCOMM Computer Communication Review* 35.4 (Aug. 2005), pp. 1–12. ISSN: 0146-4833. DOI: 10.1145/1090191.1080094.

[Gur+15] A. Gurfinkel, T. Kahsai, A. Komuravelli, and J. A. Navas. "The SeaHorn verification framework". In: *International Conference on Computer Aided Verification*. Springer. 2015, pp. 343–361.

[GW02a] T. G. Griffin and G. Wilfong. "Analysis of the MED oscillation problem in BGP". In: *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* IEEE. 2002, pp. 90–99.

[GW02b] T. G. Griffin and G. Wilfong. "On the correctness of IBGP configuration". In: *ACM SIGCOMM Computer Communication Review* 32.4 (2002), pp. 17–29.

[GW99] T. G. Griffin and G. Wilfong. "An analysis of BGP convergence properties". In: *ACM SIGCOMM Computer Communication Review* 29.4 (1999), pp. 277–288.

[Haa+17] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien. "Bringing the web up to speed with WebAssembly". In: *ACM SIGPLAN Notices* 52.6 (June 2017), pp. 185–200. ISSN: 0362-1340. DOI: 10.1145/3140587.3062363.

[Han+05] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov. "Designing extensible IP router software". In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. 2005, pp. 189–202.

[Has+] S. Hasan, P. Lapukhov, A. Madan, and O. Baldonado. "Open/R: Open routing for modern networks". https://code.fb.com/connectivity/open-r-open-routing-for-modern-networks/.

[Has15] HashiCorp. *Vault Project*. 2015. URL: https://www.vaultproject.io/ (visited on 09/12/2023).

[HB11] G. Huston and R. Bush. "Securing BGP with BGPSec". In: *The Internet Protocol Forum*. Vol. 14. 2. 2011.

[HD98] B. Hinden and D. S. E. Deering. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460. Dec. 1998. DOI: 10.17487/RFC2460.

[HDM22] C. Huitema, S. Dickinson, and A. Mankin. *DNS over Dedicated QUIC Connections*. RFC 9250. May 2022. DOI: 10.17487/RFC9250.

[Hef98] A. Heffernan. *Protection of BGP Sessions via the TCP MD5 Signature Option*. RFC 2385. Aug. 1998. DOI: 10.17487/RFC2385.

[Hei+17] J. Heitz, J. Snijders, K. Patel, I. Bagdonas, and N. Hilliard. *BGP Large Communities Attribute*. RFC 8092. Feb. 2017. DOI: 10.17487/RFC8092.

[HH06] J. Haas and S. Hares. *Definitions of Managed Objects for BGP-4*. RFC 4273. Jan. 2006. DOI: 10.17487/RFC4273.

[HHK03] M. Handley, O. Hodson, and E. Kohler. "XORP: An open platform for network research". In: *ACM SIGCOMM Computer Communication Review* 33.1 (Jan. 2003), pp. 53–57. ISSN: 0146-4833. DOI: 10.1145/774763.774771.

[HM12] G. Huston and G. G. Michaelson. *Validation of Route Origination Using the Resource Certificate Public Key Infrastructure (PKI) and Route Origin Authorizations (ROAs)*. RFC 6483. Feb. 2012. DOI: 10.17487/RFC6483.

[Høi+18] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller. "The EXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel". In: *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '18. Heraklion, Greece: Association for Computing Machinery, 2018, pp. 54–66. ISBN: 9781450360807. DOI: 10.1145/3281411.3281443.

[Hol+19] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever. "Blink: Fast Connectivity Recovery Entirely in the Data Plane". In: *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 161–176. ISBN: 978-1-931971-49-2.

[Hon+13] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. "Achieving high utilization with software-driven WAN". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. 2013, pp. 15–26.

[Hop00] C. Hopps. *Analysis of an Equal-Cost Multi-Path Algorithm*. RFC 2992. Nov. 2000. DOI: 10.17487/RFC2992.

[Hou09] R. Housley. *Cryptographic Message Syntax (CMS)*. RFC 5652. Sept. 2009. DOI: 10.17487/RFC5652.

[HPS04] Y.-C. Hu, A. Perrig, and M. Sirbu. "SPV: Secure Path Vector Routing for Securing BGP". In: *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '04. Portland, Oregon, USA: Association for Computing Machinery, 2004, pp. 179–192. ISBN: 1581138628. DOI: 10.1145/1015467.1015488.

[HS99] M. Holdrege and P. Srisuresh. *IP Network Address Translator (NAT) Terminology and Considerations*. RFC 2663. Aug. 1999. DOI: 10.17487/RFC2663.

[Hui22] C. Huitema. *Minimal implementation of the QUIC protocol*. https://github.com/private-octopus/picoquic. 2022.

[Hus23a] G. Huston. *Growth of the BGP Table - 1994 to Present*. 2023. URL: https://bgp.potaroo.net/ (visited on 05/25/2023).

[Hus23b] G. Huston. *The 32-bit AS Number Report*. 2023. URL: https://www.potaroo.net/tools/asn32/ (visited on 09/13/2023).

[Hus98] G. Huston. *ISP survival guide: strategies for running a competitive ISP*. John Wiley & Sons, Inc., 1998.

[Ier16] R. Ierusalimschy. *Programming in Lua, Fourth Edition*. Lua.Org, 2016. ISBN: 8590379868.

[IET] IETF QUIC Working Group. "QUIC implementations". https://github.com/quicwg/base-drafts/wiki/Implementations.

[Inf81] Information Sciences Institute University of Southern California. *Internet Protocol*. RFC 791. Sept. 1981. DOI: 10.17487/RFC0791.

[IO 18] IO Visor Project. "Userspace eBPF VM". https://github.com/iovisor/ubpf. 2018.

[Ish+] K. Ishiguro, T. Takada, Y. Ohara, A. D. Zinin, G. Natapov, and A. Mizutani. *Quagga software routing suite*. https://www.nongnu.org/quagga/.

[ISO] ISO. ""Intermediate system to Intermediate system intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode Network Service (ISO 8473)"". ISO/IEC 10589:2002.

[Iso17] Isovalent, Inc. "eBPF-based Networking, Observability, Security". https://cilium.io/. 2017.

[IT21] J. Iyengar and M. Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000.

[Jab15] P. Jabloner. *The two-napkin protocol*. Mar. 2015.

[Jad+22] M. Jadin, Q. De Coninck, L. Navarre, M. Schapira, and O. Bonaventure. "Leveraging eBPF to Make TCP Path-Aware". In: *IEEE Transactions on Network and Service Management* 19.3 (2022), pp. 2827–2838. DOI: 10.1109/TNSM.2022.3174138.

[Jae+23] B. Jaeger, J. Zirngibl, M. Kempf, K. Ploch, and G. Carle. "QUIC on the Highway: Evaluating Performance on High-rate Links". In: *2023 IFIP Networking Conference (IFIP Networking)*. 2023, pp. 1–9.

[Jai+13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. "B4: Experience with a globally-deployed software defined WAN". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 3–14.

[Jas+16] E. Jasinska, N. Hilliard, R. Raszuk, and N. Bakker. *Internet Exchange BGP Route Server*. RFC 7947. Sept. 2016. DOI: 10.17487/RFC7947.

[Jet+21] M. Jethanandani, K. Patel, S. Hares, and J. Haas. *BGP YANG Model for Service Provider Networks*. Internet-Draft draft-ietf-idr-bgp-model-11. Work in Progress. Internet Engineering Task Force, July 2021. 153 pp.

[Jia+23a] J. Jia, R. Sahu, A. Oswald, D. Williams, M. V. Le, and T. Xu. "Kernel Extension Verification is Untenable". In: *Proceedings of the 19th Workshop on Hot Topics in Operating Systems*. HOTOS '23. Providence, RI, USA: Association for Computing Machinery, 2023, pp. 150–157. ISBN: 9798400701955. DOI: 10.1145/3593856.3595892.

[Jia+23b] J. Jia, Y. Zhu, D. Williams, A. Arcangeli, C. Canella, H. Franke, T. Feldman-Fitzthum, D. Skarlatos, D. Gruss, and T. Xu. *Programmable System Call Security with eBPF*. 2023. arXiv: 2302.10366 [cs.OS].

[Joy+06] D. Joyal, F. Baker, S. Giacalone, D. Joyal, and P. Galecki. *OSPF Version 2 Management Information Base*. RFC 4750. Dec. 2006. DOI: 10.17487/RFC4750.

[Jun+17] R. Jung, J.-H. Jourdan, R. Krebbers, and D. Dreyer. "RustBelt: Securing the Foundations of the Rust Programming Language". In: *Proc. ACM Program. Lang.* 2.POPL (Dec. 2017). DOI: 10.1145/3158154.

[Jun21] Junos. "Junos PyEZ Developer Guide". https://www.juniper.net/documentation/en_US/junos-pyez/information-products/pathway-pages/junos-pyez-developer-guide.html. July 2021.

[Jun22] Juniper Networks. "IP Security for BGP". https://www.juniper.net/documentation/us/en/software/junos/bgp/topics/topic-map/ip_security.html. Mar. 2022.

[KaF18] M. KaFai Lau. "bpf: btf: Introduce BPF Type Format (BTF)". https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=69b693f0aefa0ed521e8bd02260523b5ae446ad7. Apr. 2018.

[Kat+12] E. Katz-Bassett, C. Scott, D. R. Choffnes, Í. Cunha, V. Valancius, N. Feamster, H. V. Madhyastha, T. Anderson, and A. Krishnamurthy. "LIFEGUARD: Practical Repair of Persistent Route Failures". In: *SIGCOMM Comput. Commun. Rev.* 42.4 (Aug. 2012), pp. 395–406. ISSN: 0146-4833. DOI: 10.1145/2377677.2377756.

[Kau05] C. Kaufman. *Internet Key Exchange (IKEv2) Protocol*. RFC 4306. Dec. 2005. DOI: 10.17487/RFC4306.

[KBC97] D. H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. Feb. 1997. DOI: 10.17487/RFC2104.

[KBS21] T. Krenc, R. Beverly, and G. Smaragdakis. "AS-level BGP community usage classification". In: *Proceedings of the 21st ACM Internet Measurement Conference*. 2021, pp. 577–592.

[Ken05a] S. Kent. *IP Authentication Header*. RFC 4302. Dec. 2005. DOI: 10.17487/RFC4302.

[Ken05b] S. Kent. *IP Encapsulating Security Payload (ESP)*. RFC 4303. Dec. 2005. DOI: 10.17487/RFC4303.

[KFR06] J. Karlin, S. Forrest, and J. Rexford. "Pretty Good BGP: Improving BGP by Cautiously Adopting Routes". In: *Proceedings of the 2006 IEEE International Conference on Network Protocols*. 2006, pp. 290–299. DOI: 10.1109/ICNP.2006.320179.

[KKC12] K. Kompella, B. Kothari, and R. Cherukuri. *Layer 2 Virtual Private Networks Using BGP for Auto-Discovery and Signaling*. RFC 6624. May 2012. DOI: 10.17487/RFC6624.

[KLS00] S. Kent, C. Lynn, and K. Seo. "Secure Border Gateway Protocol (S-BGP)". In: *IEEE Journal on Selected Areas in Communications* 18.4 (2000), pp. 582–592. DOI: 10.1109/49.839934.

[Kni+11] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. "The internet topology zoo". In: *IEEE Journal on Selected Areas in Communications* 29.9 (2011), pp. 1765–1775.

[Kos+22] M. Kosek, T. V. Doan, M. Granderath, and V. Bajpai. "One to Rule Them All? A First Look at DNS over QUIC". In: *Passive and Active Measurement: 23rd International Conference, PAM 2022, Virtual Event, March 28–30, 2022, Proceedings*. Springer. 2022, pp. 537–551.

[Kre+14] D. Kreutz, F. M. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. "Software-defined networking: A comprehensive survey". In: *Proceedings of the IEEE* 103.1 (2014), pp. 14–76.

[KT14] D. Kroening and M. Tautschnig. "CBMC–C bounded model checker". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2014, pp. 389–391.

[KW10] D. Katz and D. Ward. *Bidirectional Forwarding Detection (BFD)*. RFC 5880. June 2010. DOI: 10.17487/RFC5880.

[Lan+17] A. Langley, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, A. Riddoch, W.-T. Chang, Z. Shi, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, and I. Swett. "The QUIC Transport Protocol: Design and Internet-Scale Deployment". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '17*. Los Angeles, CA, USA: ACM Press, 2017, pp. 183–196. ISBN: 978-1-4503-4653-5. DOI: 10.1145/3098822.3098842. (Visited on 11/08/2019).

[LCT96] T. Li, R. Chandra, and P. S. Traina. *BGP Communities Attribute*. RFC 1997. Aug. 1996. DOI: 10.17487/RFC1997.

[LDD14] A. Lodhi, A. Dhamdhere, and C. Dovrolis. "Open peering by Internet transit providers: Peer preference or peer pressure?" In: *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE. 2014, pp. 2562–2570.

[LGS13] R. Lychev, S. Goldberg, and M. Schapira. "BGP Security in Partial Deployment: Is the Juice Worth the Squeeze?" In: *SIGCOMM Comput. Commun. Rev.* SIGCOMM '13 43.4 (Aug. 2013), pp. 171–182. ISSN: 0146-4833. DOI: 10.1145/2534169.2486010.

[LHZ14] Q. Li, Y.-C. Hu, and X. Zhang. "Even rockets cannot make pigs fly sustainably: Can BGP be secured with BGPsec". In: *Workshop SENT'14, 23 February 2014, San Diego, USA, Copyright 2014 Internet Society: Proceedings*. Internet Society. 2014.

[Li+15] Q. Li, X. Zhang, X. Zhang, and P. Su. "Invalidating Idealized BGP Security Proposals and Countermeasures". In: *IEEE Transactions on Dependable and Secure Computing* 12.3 (2015), pp. 298–311. DOI: 10.1109/TDSC.2014.2345381.

[Lin+18] A. Lindem, A. Roy, D. Goethals, V. R. Vallem, and F. Baker. *OSPFv3 Link State Advertisement (LSA) Extensibility*. RFC 8362. Apr. 2018. DOI: 10.17487/RFC8362.

[Lin22] Linux User's Manual. *bpf-helpers(7)*. Sept. 2022. URL: https://man7.org/linux/man-pages/man7/bpf-helpers.7.html (visited on 09/13/2023).

[Liu+23] Y. Liu, Y. Ma, Q. D. Coninck, O. Bonaventure, C. Huitema, and M. Kühlewind. *Multipath Extension for QUIC*. Internet-Draft draft-ietf-quic-multipath-04. Work in Progress. Internet Engineering Task Force, Mar. 2023. 30 pp.

[LK12] M. Lepinski and S. Kent. *An Infrastructure to Support Secure Internet Routing*. RFC 6480. Feb. 2012. DOI: 10.17487/RFC6480.

[LKK12] M. Lepinski, D. Kong, and S. Kent. *A Profile for Route Origin Authorizations (ROAs)*. RFC 6482. Feb. 2012. DOI: 10.17487/RFC6482.

[Lod+14] A. Lodhi, N. Larson, A. Dhamdhere, C. Dovrolis, and K. Claffy. "Using peeringDB to understand the peering ecosystem". In: *ACM SIGCOMM Computer Communication Review* 44.2 (2014), pp. 20–27.

[LPM16] P. Lapukhov, A. Premji, and J. Mitchell. *Use of BGP for Routing in Large-Scale Data Centers*. RFC 7938. Aug. 2016. DOI: 10.17487/RFC7938.

[LR89] K. Lougheed and Y. Rekhter. *Border Gateway Protocol (BGP)*. RFC 1105. June 1989. DOI: 10.17487/RFC1105.

[LS16] J. Looney and S. Smith. *Automating Junos Administration: Doing More with Less*. " O'Reilly Media, Inc.", 2016.

[LS17] M. Lepinski and K. Sriram. *BGPsec Protocol Specification*. RFC 8205. Sept. 2017. DOI: 10.17487/RFC8205.

[LT21] P. Lapukhov and J. Tantsura. *Equal-Cost Multipath Considerations for BGP*. Internet-Draft draft-lapukhov-bgp-ecmp-considerations-07. Work in Progress. Internet Engineering Task Force, June 2021. 7 pp.

[Luc+19] M. Luckie, R. Beverly, R. Koga, K. Keys, J. A. Kroll, and K. Claffy. "Network hygiene, incentives, and regulation: deployment of source address validation in the Internet". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 465–480.

[LY06] C. M. Lonvick and T. Ylonen. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. Jan. 2006. DOI: 10.17487/RFC4253.

[Lyn99] C. Lynn. *X.509 Extensions for Authorization of IP Addresses, AS Numbers, and Routers within an AS*. Internet-Draft draft-clynn-bgp-x509-auth-00. Work in Progress. Internet Engineering Task Force, June 1999. 13 pp.

[Mal98] G. S. Malkin. *RIP Version 2*. RFC 2453. Nov. 1998. DOI: 10.17487/RFC2453.

[Man+09] V. Manral, M. Fanto, T. Li, R. White, R. Atkinson, and M. Bhatia. *IS-IS Generic Cryptographic Authentication*. RFC 5310. Feb. 2009. DOI: 10.17487/RFC5310.

[Mao+03] Z. M. Mao, R. Bush, T. G. Griffin, and M. Roughan. "BGP Beacons". In: *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*. IMC '03. Miami Beach, FL, USA: Association for Computing Machinery, 2003, pp. 1–14. ISBN: 1581137737. DOI: 10.1145/948205.948207.

[Mat21] M. Matějka. *BIRD Journey to Threads. Chapter 0: The Reason Why*. Mar. 2021. URL: https://en.blog.nic.cz/2021/03/15/bird-journey-to-threads-chapter-0-the-reason-why/ (visited on 09/14/2023).

[McK+08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: enabling innovation in campus networks". In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[McQ+21] S. McQuistin, M. Karan, P. Khare, C. Perkins, G. Tyson, M. Purver, P. Healey, W. Iqbal, J. Qadir, and I. Castro. "Characterising the IETF through the Lens of RFC Deployment". In: *Proceedings of the 21st ACM Internet Measurement Conference*. IMC '21. Virtual Event: Association for Computing Machinery, 2021, pp. 137–149. ISBN: 9781450391290. DOI: 10.1145/3487552.3487821.

[Mia+18] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal. "Creating complex network service with ebpf: Experience and lessons learned". In: *High Performance Switching and Routing (HPSR). IEEE* (2018).

[Mica] Microsoft Azure. *Software for open networking in the cloud*. URL: https://sonicfoundation.dev/ (visited on 06/19/2023).

[Micb] Microsoft Corporation. *eBPF for Windows*. URL: https://github.com/microsoft/ebpf-for-windows (visited on 06/22/2023).

[Mic19] Microsoft. *MsQuic: Cross-platform, C implementation of the IETF QUIC protocol, exposed to C, C++, C# and Rust*. 2019. URL: https://github.com/microsoft/msquic (visited on 06/26/2023).

[MJ93] S. McCanne and V. Jacobson. "The BSD Packet Filter: A New Architecture for User-Level Packet Capture". In: *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*. USENIX'93. San Diego, California: USENIX Association, 1993, p. 2.

[MK14] N. D. Matsakis and F. S. Klock. "The Rust Language". In: *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*. HILT '14. Portland, Oregon, USA: Association for Computing Machinery, 2014, pp. 103–104. ISBN: 9781450332170. DOI: 10.1145/2663171.2663188.

[MK20] D. Ma and S. Kent. *Requirements for Resource Public Key Infrastructure (RPKI) Relying Parties*. RFC 8897. Sept. 2020. DOI: 10.17487/RFC8897.

[MMD22] E. Marechal, P. Mérindol, and B. Donnet. "ISP Probing Reduction with Anaximander". In: *Passive and Active Measurement*. Ed. by O. Hohlfeld, G. Moura, and C. Pelsser. Cham: Springer International Publishing, 2022, pp. 441–469. ISBN: 978-3-030-98785-5.

[Moy91] J. Moy. *OSPF Version 2*. RFC 1247. July 1991. DOI: 10.17487/RFC1247.

[Moy98] J. Moy. *OSPF Version 2*. RFC 2328. Apr. 1998. DOI: 10.17487/RFC2328.

[MP19] L. Miller and C. Pelsser. "A Taxonomy of Attacks Using BGP Blackholing". In: *Computer Security – ESORICS 2019*. Ed. by K. Sako, S. Schneider, and P. Y. A. Ryan. Cham: Springer International Publishing, 2019, pp. 107–127. ISBN: 978-3-030-29959-0.

[MPE18] A. Mitseva, A. Panchenko, and T. Engel. "The state of affairs in BGP security: A survey of attacks and defenses". In: *Computer Communications* 124 (2018), pp. 45–60. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2018.04.013.

[Mur06] S. L. Murphy. *BGP Security Vulnerabilities Analysis*. RFC 4272. Jan. 2006. DOI: 10.17487/RFC4272.

[MWA02] R. Mahajan, D. Wetherall, and T. Anderson. "Understanding BGP Misconfiguration". In: *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. Vol. 32. SIGCOMM '02 4. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, Aug. 2002, pp. 3–16. ISBN: 158113570X. DOI: 10.1145/633025.633027.

[Nak+22] R. Nakamura, K. Shimizu, T. Kamata, and C. Pelsser. "A First Measurement with BGP Egress Peer Engineering". In: *Passive and Active Measurement*. Ed. by O. Hohlfeld, G. Moura, and C. Pelsser. Cham: Springer International Publishing, 2022, pp. 199–215. ISBN: 978-3-030-98785-5.

[Naw+19] M. Nawrocki, J. Blendin, C. Dietzel, T. C. Schmidt, and M. Wählisch. "Down the black hole: dismantling operational practices of BGP blackholing at IXPs". In: *Proceedings of the Internet Measurement conference*. 2019, pp. 435–448.

[Nem+21] E. N. Nemmi, F. Sassi, M. La Morgia, C. Testart, A. Mei, and A. Dainotti. "The parallel lives of Autonomous Systems: ASN Allocations vs. BGP". In: *Proceedings of the 21st ACM Internet Measurement Conference*. 2021, pp. 593–611.

[Ong+21] P. Ongkanchana, R. Fontugne, H. Esaki, J. Snijders, and E. Aben. "Hunting BGP Zombies in the Wild". In: *Proceedings of the Applied Networking Research Workshop*. ANRW '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 1–7. ISBN: 9781450386180. DOI: 10.1145/3472305.3472315.

[Ora15] Oracle. *Oracle Key Vault*. 2015. URL: https://www.oracle.com/security/database-security/key-vault/ (visited on 09/12/2023).

[Pad+19] R. Padmanabhan, A. Schulman, D. Levin, and N. Spring. "Residential Links under the Weather". In: *Proceedings of the ACM Special Interest Group on Data Communication*. SIGCOMM '19. Beijing, China: Association for Computing Machinery, 2019, pp. 145–158. ISBN: 9781450359566. DOI: 10.1145/3341302.3342084.

[Pan+19] M. Panchenko, R. Auler, B. Nell, and G. Ottoni. "Bolt: a practical binary optimizer for data centers and beyond". In: *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*. IEEE Press. 2019, pp. 2–14.

[Pat+03] P. Patel, A. Whitaker, D. Wetherall, J. Lepreau, and T. Stack. "Upgrading Transport Protocols using Untrusted Mobile Code". In: *ACM SIGOPS Operating Systems Review* 37.5 (2003), pp. 1–14.

[Pat+23] K. Patel, A. Lindem, S. Zandi, and W. Henderickx. *BGP Link-State Shortest Path First (SPF) Routing*. Internet-Draft draft-ietf-lsvr-bgp-spf-22. Work in Progress. Internet Engineering Task Force, Mar. 2023. 38 pp.

[Pau+23] T. Pauly, B. Trammell, A. Brunstrom, G. Fairhurst, and C. Perkins. *An Architecture for Transport Services*. Internet-Draft draft-ietf-taps-arch-17. Work in Progress. Internet Engineering Task Force, Mar. 2023. 32 pp.

[PB22] M. Piraux and O. Bonaventure. *Additional addresses for QUIC*. Internet-Draft draft-piraux-quic-additional-addresses-00. Work in Progress. Internet Engineering Task Force, Oct. 2022. 7 pp.

[PCV14] K. Patel, E. Chen, and B. Venkatachalapathy. *Enhanced Route Refresh Capability for BGP-4*. RFC 7313. July 2014. DOI: 10.17487/RFC7313.

[Pig+07] C. Pignataro, P. Savola, D. Meyer, V. Gill, and J. Heasley. *The Generalized TTL Security Mechanism (GTSM)*. RFC 5082. Oct. 2007. DOI: 10.17487/RFC5082.

[Pos80] J. Postel. *User Datagram Protocol*. RFC 768. Aug. 1980. DOI: 10.17487/RFC0768.

[Pos81] J. Postel. *Transmission Control Protocol*. RFC 793. Sept. 1981. DOI: 10.17487/RFC0793.

[PR83] J. Poster and J. Reynolds. *Telnet Protocol Specification*. RFC 854. May 1983. DOI: 10.17487/RFC0854.

[Prz+23] T. Przygienda, A. Sharma, P. Thubert, B. Rijsman, D. Afanasiev, and J. Head. *RIFT: Routing in Fat Trees*. Internet-Draft draft-ietf-rift-rift-17. Work in Progress. Internet Engineering Task Force, Mar. 2023. 174 pp.

[Pse+23] P. Psenak, S. Hegde, C. Filsfils, K. Talaulikar, and A. Gulko. *IGP Flexible Algorithm*. RFC 9350. Feb. 2023. DOI: 10.17487/RFC9350.

[QHP13] L. Quan, J. Heidemann, and Y. Pradkin. "Trinocular: Understanding Internet Reliability through Adaptive Probing". In: *SIGCOMM Comput. Commun. Rev.* SIGCOMM '13 43.4 (Aug. 2013), pp. 255–266. ISSN: 0146-4833. DOI: 10.1145/2534169.2486017.

[Qu+21] Y. Qu, J. Tantsura, A. Lindem, and X. Liu. *A YANG Data Model for Routing Policy*. Internet-Draft draft-ietf-rtgwg-policy-model-31. Work in Progress. Internet Engineering Task Force, Aug. 2021. 42 pp.

[Quo+03] B. Quoitin, C. Pelsser, L. Swinnen, O. Bonaventure, and S. Uhlig. "Interdomain traffic engineering with BGP". In: *IEEE Communications magazine* 41.5 (2003), pp. 122–128.

[Ras+18] R. Raszuk et al. "BGP Community Container Attribute". Internet draft, draft-ietf-idr-wide-bgp-communities-05, work in progress. July 2018.

[RB22] N. Rybowski and O. Bonaventure. "Evaluating OSPF Convergence with Ns-3 DCE". In: *Proceedings of the 2022 Workshop on Ns-3*. WNS3 '22. Virtual Event, USA: Association for Computing Machinery, 2022, pp. 120–126. ISBN: 9781450396516. DOI: 10.1145/3532577.3532597.

[Rek+07] Y. Rekhter, J. Scudder, S. R. Sangli, E. Chen, and R. Fernando. *Graceful Restart Mechanism for BGP*. RFC 4724. Jan. 2007. DOI: 10.17487/RFC4724.

[Res18] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446.

[Ret+23] A. Retana, Y. Qu, J. Haas, S. Chen, and J. Tantsura. *BGP over QUIC*. Internet-Draft draft-retana-idr-bgp-quic-01. Work in Progress. Internet Engineering Task Force, Mar. 2023. 21 pp.

[Reu+18] A. Reuter, R. Bush, I. Cunha, E. Katz-Bassett, T. C. Schmidt, and M. Wählisch. "Towards a rigorous methodology for measuring adoption of RPKI route validation and filtering". In: *ACM SIGCOMM Computer Communication Review* 48.1 (2018), pp. 19–27.

[RHL06] Y. Rekhter, S. Hares, and T. Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. Jan. 2006. DOI: 10.17487/RFC4271.

[RIP10] RIPE Labs. "RIPE NCC and Duke University BGP Experiment". https://morse.colorado.edu/~epperson/courses/routing-protocols/handouts/ripe-bgp-experiment-gone-awry.pdf. Aug. 2010.

[RIP23] RIPE NCC. 2023. URL: https://atlas.ripe.net/ (visited on 09/18/2023).

[RL94] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 1654. July 1994. DOI: 10.17487/RFC1654.

[RL95] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 1771. Mar. 1995. DOI: 10.17487/RFC1771.

[RM12] E. Rescorla and N. Modadugu. *Datagram Transport Layer Security Version 1.2*. RFC 6347. Jan. 2012. DOI: 10.17487/RFC6347.

[Ros13] J. Roskind. June 2013. URL: https://blog.chromium.org/2013/06/experimenting-with-quic.html (visited on 06/26/2023).

[RQT22] A. Retana, Y. Qu, and J. Tantsura. *Use of Streams in BGP over QUIC*. Internet-Draft draft-retana-idr-bgp-quic-stream-02. Work in Progress. Internet Engineering Task Force, May 2022. 11 pp.

[RR06] Y. Rekhter and E. C. Rosen. *BGP/MPLS IP Virtual Private Networks (VPNs)*. RFC 4364. Feb. 2006. DOI: 10.17487/RFC4364.

[Ryb+21] N. Rybowski, Q. De Coninck, T. Rousseaux, A. Legay, and O. Bonaventure. "Implementing the Plugin Distribution System". In: *Proceedings of the SIGCOMM '21 Poster and Demo Sessions*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 39–41. ISBN: 9781450386296.

[SA23] K. Sriram and A. Azimov. *Methods for Detection and Mitigation of BGP Route Leaks*. Internet-Draft draft-ietf-grow-route-leak-detection-mitigation-09. Work in Progress. Internet Engineering Task Force, July 2023. 10 pp.

[SAD22] H. Sharaf, I. Ahmad, and T. Dimitriou. "Extended Berkeley Packet Filter: An Application Perspective". In: *IEEE Access* 10 (2022), pp. 126370–126393. DOI: 10.1109/ACCESS.2022.3226269.

[Sal+21] L. Salamatian, F. Douzet, K. Salamatian, and K. Limonier. "The geopolitics behind the routes data travel: a case study of Iran". In: *Journal of Cybersecurity* 7.1 (Aug. 2021). tyab018. ISSN: 2057-2085. DOI: 10.1093/cybsec/tyab018. eprint: https://academic.oup.com/cybersecurity/article-pdf/7/1/tyab018/39765655/tyab018.pdf.

[SAM22] J. Snijders, M. Abrahamsson, and B. Maddison. *Resource Public Key Infrastructure (RPKI) object profile for Discard Origin Authorizations (DOA)*. Internet-Draft draft-spaghetti-sidrops-rpki-doa-00. Work in Progress. Internet Engineering Task Force, Mar. 2022. 13 pp.

[San+13] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and D. C. Adams. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. RFC 6960. June 2013. DOI: 10.17487/RFC6960.

[Sav+16] D. Savage, J. Ng, S. Moore, D. Slice, P. Paluch, and R. White. *Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)*. RFC 7868. May 2016. DOI: 10.17487/RFC7868.

[SB10] M. Shand and S. Bryant. *IP Fast Reroute Framework*. RFC 5714. Jan. 2010. DOI: 10.17487/RFC5714.

[SBV21] T. Schneider, R. Birkner, and L. Vanbever. "Snowcap: Synthesizing Network-Wide Configuration Updates". In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. SIGCOMM '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 33–49. ISBN: 9781450383837. DOI: 10.1145/3452296.3472915.

[Sch+23] T. Schneider, R. Schmid, S. Vissicchio, and L. Vanbever. "Taming the Transient While Reconfiguring BGP". In: *Proceedings of the ACM SIGCOMM 2023 Conference*. ACM SIGCOMM '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 77–93. ISBN: 9798400702365. DOI: 10.1145/3603269.3604855.

[Ser+18] P. Sermpezis, V. Kotronis, A. Dainotti, and X. Dimitropoulos. "A survey among network operators on BGP prefix hijacking". In: *ACM SIGCOMM Computer Communication Review* 48.1 (2018), pp. 64–69.

[SFS16] J. Scudder, R. Fernando, and S. Stuart. *BGP Monitoring Protocol (BMP)*. RFC 7854. June 2016. DOI: 10.17487/RFC7854.

[SG96] B. Smith and J. Garcia-Luna-Aceves. "Securing the border gateway routing protocol". In: *Proceedings of GLOBECOM'96. 1996 IEEE Global Telecommunications Conference*. Vol. MiniConfInternet. 1996, pp. 81–85. DOI: 10.1109/GLOCOM.1996.586129.

[Sha+11] R. Shakir, R. Raszuk, R. Shakir, and D. Freedman. *BGP OPERATIONAL Message*. Internet-Draft draft-frs-bgp-operational-message-00. Work in Progress. Internet Engineering Task Force, July 2011. 26 pp.

[Shr+21] T. Shreedhar, R. Panda, S. Podanev, and V. Bajpai. "Evaluating QUIC Performance Over Web, Cloud Storage, and Video Workloads". In: *IEEE Transactions on Network and Service Management* 19.2 (2021), pp. 1366–1381.

[Sin+21] R. Singh, M. Mukhtar, A. Krishna, A. Parkhi, J. Padhye, and D. Maltz. "Surviving switch failures in cloud datacenters". In: *ACM SIGCOMM Computer Communication Review* 51.2 (2021), pp. 2–9.

[SK05] K. Seo and S. Kent. *Security Architecture for the Internet Protocol*. RFC 4301. Dec. 2005. DOI: 10.17487/RFC4301.

[Spr+04] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. "Measuring ISP topologies with Rocketfuel". In: *IEEE/ACM Transactions on Networking* 12.1 (2004), pp. 2–16. DOI: 10.1109/TNET.2003.822655.

[Sri+16] K. Sriram, D. Montgomery, D. R. McPherson, E. Osterweil, and B. Dickson. *Problem Definition and Classification of BGP Route Leaks*. RFC 7908. June 2016. DOI: 10.17487/RFC7908.

[SsA20] J. Snijders, stucchi-listsglevia.com, and M. Aelmans. *RPKI Autonomous Systems Cones: A Profile To Define Sets of Autonomous Systems Numbers To Facilitate BGP Filtering*. Internet-Draft draft-ietf-grow-rpki-as-cones-02. Work in Progress. Internet Engineering Task Force, Apr. 2020. 10 pp.

[SSV22] T. Schneider, R. Schmid, and L. Vanbever. "On the Complexity of Network-Wide Configuration Synthesis". In: *2022 IEEE 30th International Conference on Network Protocols (ICNP)*. Oct. 2022, pp. 1–11. DOI: 10.1109/ICNP55882.2022.9940325.

[Sta14a] A. Starovoitov. "Rework/Optimize internal BPF interpreter's instruction set". https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=bd4cf0ed331a275e9bf5a49e6d0fd55dffc551b8. Mar. 2014.

[Sta14b] A. Starovoitov. "The Linux kernel static checker". https://github.com/torvalds/linux/blob/master/kernel/bpf/verifier.c. Sept. 2014.

[Sta15] A. Starovoitov. "BPF-in-kernel virtual machine". In: *Linux Kernel Developers' Netconf* (2015).

[Sta18] A. Starovoitov. "bpf: introduce BPF_RAW_TRACEPOINT". https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=c4f6699dfcb8558d138fe838f741b2c10f416cf9. Mar. 2018.

[Ste06] R. Steenbergen. "IRR Power Tools - A utility for managing Internet Routing Registry (IRR) filters". Presented at NANOG36 https://archive.nanog.org/meetings/nanog36/presentations/steenbergen.pdf. 2006.

[Ste07] R. R. Stewart. *Stream Control Transmission Protocol*. RFC 4960. Sept. 2007. DOI: 10.17487/RFC4960.

[Str+18] F. Streibelt, F. Lichtblau, R. Beverly, A. Feldmann, C. Pelsser, G. Smaragdakis, and R. Bush. "BGP Communities: Even more Worms in the Routing Can". In: *Proceedings of the Internet Measurement Conference 2018*. IMC '18. ACM. Boston, MA, USA: Association for Computing Machinery, 2018, pp. 279–292. ISBN: 978-1-4503-5619-0. DOI: 10.1145/3278532.3278557.

[SV18] H. Smit and G. V. de Velde. *IS-IS Flooding over TCP*. Internet-Draft draft-hsmit-lsr-isis-flooding-over-tcp-00. Work in Progress. Internet Engineering Task Force, Oct. 2018. 14 pp.

[SVG16] Y. Song, A. Venkataramani, and L. Gao. "Identifying and Addressing Reachability and Policy Attacks in 'Secure' BGP". In: *IEEE/ACM Transactions on Networking* 24.5 (2016), pp. 2969–2982. ISSN: 10636692. DOI: 10.1109/TNET.2015.2503642.

[TB19] V.-H. Tran and O. Bonaventure. "Beyond socket options: making the Linux TCP stack truly extensible". In: *2019 IFIP Networking Conference (IFIP Networking)*. IEEE. 2019, pp. 1–9. DOI: 10.23919/IFIPNetworking46909.2019.8999401.

[TBM10] D. J. D. Touch, R. Bonica, and A. J. Mankin. *The TCP Authentication Option*. RFC 5925. June 2010. DOI: 10.17487/RFC5925.

[Tei+04] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford. "Dynamics of Hot-Potato Routing in IP Networks". In: *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '04/Performance '04. New York, NY, USA: Association for Computing Machinery, 2004, pp. 307–319. ISBN: 1581138733. DOI: 10.1145/1005686.1005723.

[Ten+97] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. "A survey of active network research". In: *IEEE communications Magazine* 35.1 (1997), pp. 80–86.

[Tes+19] C. Testart, P. Richter, A. King, A. Dainotti, and D. Clark. "Profiling BGP serial hijackers: capturing persistent misbehavior in the global routing table". In: *Proceedings of the Internet Measurement Conference*. 2019, pp. 420–434.

[The] The OpenBSD Project. "OpenBGPD". http://openbgpd.com/.

[The03] The OpenSSL Project. "OpenSSL: The Open Source toolkit for SSL/TLS". www.openssl.org. Apr. 2003.

[TM96] A. Trigell and P. Mackerras. *rsync: An open source utility that provides fast incremental file transfer*. June 1996.

[TSR06] D. Tappan, S. R. Sangli, and Y. Rekhter. *BGP Extended Communities Attribute*. RFC 4360. Feb. 2006. DOI: 10.17487/RFC4360.

[TUD07] R. Teixeira, S. Uhlig, and C. Diot. "BGP Route Propagation between Neighboring Domains". In: *Proceedings of the 8th International Conference on Passive and Active Network Measurement*. PAM'07. Louvain-la-Neuve, Belgium: Springer-Verlag, 2007, pp. 11–21. ISBN: 9783540716167.

[Tur04] D. Turk. *Configuring BGP to Block Denial-of-Service Attacks*. RFC 3882. Oct. 2004. DOI: 10.17487/RFC3882.

[TW02] D. Tennenhouse and D. Wetherall. "Towards an active network architecture". In: *Proceedings DARPA Active Networks Conference and Exposition*. 2002, pp. 2–15. DOI: 10.1109/DANCE.2002.1003480.

[TW07] D. L. Tennenhouse and D. J. Wetherall. "Towards an Active Network Architecture". In: 37.5 (Oct. 2007), pp. 81–94. ISSN: 0146-4833. DOI: 10.1145/1290168.1290180.

[TW96] D. L. Tennenhouse and D. J. Wetherall. "Towards an active network architecture". In: *ACM SIGCOMM Computer Communication Review* 26.2 (1996), pp. 5–17.

[Upa] G. R. Upadhaya. "Best practices for ISPs". http://www.pch.net/resources/tutorial/ispbcp.

[Van+13] Y. Vanaubel, J.-J. Pansiot, P. Mérindol, and B. Donnet. "Network fingerprinting: TTL-based router signatures". In: *Proceedings of the 2013 conference on Internet measurement conference*. 2013, pp. 369–376.

[Van09] L. Vanbever. "Customized BGP Route Selection Using BGP/MPLS VPNs". In: *Routing Symposium, Cisco Systems*. 2009.

[VC12] Q. Vohra and E. Chen. *BGP Support for Four-Octet Autonomous System (AS) Number Space*. RFC 6793. Dec. 2012. DOI: 10.17487/RFC6793.

[VCD14] S. Vissicchio, L. Cittadini, and G. Di Battista. "On iBGP routing policies". In: *IEEE/ACM Transactions on Networking* 23.1 (2014), pp. 227–240.

[VCG98] C. Villamizar, R. Chandra, and D. R. Govindan. *BGP Route Flap Damping*. RFC 2439. Nov. 1998. DOI: 10.17487/RFC2439.

[VVB14] S. Vissicchio, L. Vanbever, and O. Bonaventure. "Opportunities and Research Challenges of Hybrid Software Defined Networks". In: *SIGCOMM Comput. Commun. Rev.* 44.2 (Apr. 2014), pp. 70–75. ISSN: 0146-4833. DOI: 10.1145/2602204.2602216.

[Wäh+13] M. Wählisch, F. Holler, T. C. Schmidt, and J. H. Schiller. "RTR-lib: An Open-Source Library in C for RPKI-based Prefix Origin Validation". In: *Presented as part of the 6th Workshop on Cyber Security Experimentation and Test*. 2013.

[Wäh+15] M. Wählisch, R. Schmidt, T. C. Schmidt, O. Maennel, S. Uhlig, and G. Tyson. "RiPKI: The tragic story of RPKI deployment in the Web ecosystem". In: *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*. 2015, pp. 1–7.

[Wal+02] D. Walton et al. "Advertisement of Multiple Paths in BGP". Internet draft, draft-walton-bgp-add-paths-00, work in progress. May 2002.

[Wal+16] D. Walton, A. Retana, E. Chen, and J. Scudder. *Advertisement of Multiple Paths in BGP*. RFC 7911. July 2016. DOI: 10.17487/RFC7911.

[Was23a] Wasmer Inc. "WASIX". https://wasix.org/. May 2023.

[Was23b] Wasmer Inc. "wasix libc implementation for WebAssembly". https://github.com/wasix-org/wasix-libc. May 2023.

[WGT98] D. Wetherall, J. Guttag, and D. Tennenhouse. "ANTS: a toolkit for building and dynamically deploying network protocols". In: *1998 IEEE Open Architectures and Network Programming*. 1998, pp. 117–129. DOI: 10.1109/OPNARC.1998.662048.

[Whi03] R. White. *Deployment Considerations for Secure Origin BGP (soBGP)*. Internet-Draft draft-white-sobgp-bgp-deployment-01. Work in Progress. Internet Engineering Task Force, June 2003. 12 pp.

[Whi20] J. Whited. "CoreBGP - Plugging in to BGP". https://github.com/jwhited/corebgp. July 2020.

[Wira] T. Wirtgen. *xBGP API documentation*. https://github.com/pluginized-protocols/xbgp_plugins/blob/master/xbgp_compliant_api/xbgp_plugin_api.h.

[Wirb] T. Wirtgen. *xBGP source code*. https://github.com/pluginized-protocols/libxbgp.

[WMS04] R. White, D. McPherson, and S. Sangli. *Practical BGP*. Addison Wesley Longman Publishing Co., Inc., 2004.

[Wu+13] Q. Wu et al. "BGP attribute for North-Bound Distribution of Traffic Engineering (TE) performance Metrics". Internet draft, draft-wu-idr-te-pm-bgp, work in progress. Oct. 2013.

[WY05] X. Wang and H. Yu. "How to Break MD5 and Other Hash Functions". In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by R. Cramer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 19–35. ISBN: 978-3-540-32055-5.

[XDB18] M. Xhonneux, F. Duchene, and O. Bonaventure. "Leveraging eBPF for programmable network functions with IPv6 Segment Routing". In: *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. CoNEXT '18. ACM. Heraklion, Greece: Association for Computing Machinery, 2018, pp. 67–72. ISBN: 9781450360807. DOI: 10.1145/3281411.3281426.

[YB21] A. Yu and T. A. Benson. "Dissecting performance of production QUIC". In: *Proceedings of the Web Conference 2021*. 2021, pp. 1157–1168.

[Zha+11] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. "SCION: Scalability, Control, and Isolation on Next-Generation Networks". In: *2011 IEEE Symposium on Security and Privacy*. 2011, pp. 212–227. DOI: 10.1109/SP.2011.45.

[Zho+22] Y. Zhong, H. Li, Y. J. Wu, I. Zarkadas, J. Tao, E. Mesterhazy, M. Makris, J. Yang, A. Tai, R. Stutsman, and A. Cidon. "XRP: In-Kernel Storage Functions with eBPF". In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, July 2022, pp. 375–393. ISBN: 978-1-939133-28-1.

[Zho+23] Y. Zhou, Z. Wang, S. Dharanipragada, and M. Yu. "Electrode: Accelerating Distributed Protocols with eBPF". In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 1391–1407. ISBN: 978-1-939133-33-5.

[Zir+21] J. Zirngibl, P. Buschmann, P. Sattler, B. Jaeger, J. Aulbach, and G. Carle. "It's over 9000: Analyzing Early QUIC Deployments with the Standardization on the Horizon". In: *Proceedings of the 21st ACM Internet Measurement Conference*. IMC '21. Virtual Event: Association for Computing Machinery, 2021, pp. 261–275. ISBN: 9781450391290. DOI: 10.1145/3487552.3487826.

[ZMW07] Y. Zhang, Z. M. Mao, and J. Wang. "Low-Rate TCP-Targeted DoS Attacks Disrupts Internet Routing". In: *Proceedings of 14th Annual Network & Distributed System Security Symposium (NDSS)*. 2007.