The Role and Challenges of RPKI in Edge Node Design and BGP Hijacking Prevention

by

[Zhang Gaoxing]

A Dissertation

for the EPQ Project

[Mar 2023]

Word Count : 5295

Abstract

With the popularization of Internet applications, more and more network users want to have their own Autonomous System Number (ASN) and declare their Internet Protocol (IP) addresses through Border Gateway Protocol (BGP) to provide IP Transit services to their customers, which greatly increases the demand for the use of BGP backbone network devices with security mechanisms. this project implements a BGP-based backbone solution in a purely open-source way(BIRD), which adopts the Internet Resource Public key Infrastructure (RPKI) to achieve its security protection and make the network have functions such as anti-BGP hijacking. The solution is currently deployed on the author's network project (HUIZE-NET, https://huize.asia) node, which has been certified by Mutually Agreed Norms for Routing Security (MANRS). The backbone solution implemented with open-source software can meet the needs of individuals and small businesses to have a powerful and low-cost autonomous network system, and has the value of application promotion.

Keywords: border gateway protocol; Internet numbering resource public key infrastructure; IP address; AS number

1. Introduction

The Internet consists of numerous networks called autonomous systems, and data is transmitted between them following a routing protocol. A so-called routing protocol is an expertly designed specification that specifies how packets are forwarded. Routing protocols can be divided into two categories depending on the scope of application: routing protocols within the same Autonomous System (AS) are called Interior Gateway Protocols(IGP), and routing protocols between ASes are called External Border Gateway Protocol (EBGP) The EBGP currently uses the BGP developed by the Internet Engineering Task Force (IETF) （Griffin & Wilfong，1999）. The IETF has developed several recommendations for BGP, and the current version in use is RFC 4271, also known as BGP4 (Rekhter , Li & Hares ,2006). As a protocol commonly adopted and supported by most router manufacturers, BGP has become the de facto standard protocol for inter-domain routing on the Internet.

After the Internet service providers build the interconnection network according to the routing protocol, the network users expect the network operation to be reliable and secure. However, the BGP has its own vulnerabilities that make the Internet routing infrastructure vulnerable to attacks. In order to solve the security problem of BGP, Kent (2002), the chief scientist of security field of BBN Technologies, introduced the first BGP security solution Secure BGP (S-BGP) , whose goal was to fundamentally solve the security problem of BGP protocol by using cryptographic techniques. S-BGP mechanism is not suitable for the operation and application in real network environment. As a result, optimization schemes based on S-BGP have been emerging （White 2003；Oorschot , Wan& Kranakis, 2007）. The RPKI was developed based on the S-BGP mechanism （Lepinski & Kent，2012；Wählisch, Maennel &Schmidt, 2012）.

Since its inception, RPKI has attracted the attention of the industry, and major Internet service providers, cloud service providers, and Internet exchanges are gradually implementing RPKI as an authentication method. As a result of their own research and market demands, router hardware vendors have been releasing routers that support RPKI routing origin authentication. Additionally, the global RPKI application trend demonstrates that the development of RPKI Internet routing security infrastructure has gained support from the Internet community.

The reasons for the above: first, BGP is the de facto inter-domain routing standard protocol of the Internet, and then, the RPKI Internet routing security infrastructure has become the consensus of the Internet community. Therefore, this project uses the open-source software BIRD to implement an autonomous system based on the BGP protocol, and the system supports the RPKI routing origin verification function. The idea is in line with the trend of Internet applications and has a price advantage compared with the products of router equipment manufacturers, which can meet the needs of individuals and small enterprises to establish their own autonomous systems.

2. Literature Review

2.1 BGP protocol

The mode of BGP session can be divided into two categories: peering and IP transit (Rekhter, 2006), as shown in Figure 1, AS stands for autonomous network system. There is a difference between Peering and IP Transit in terms of commercial operation and technology.

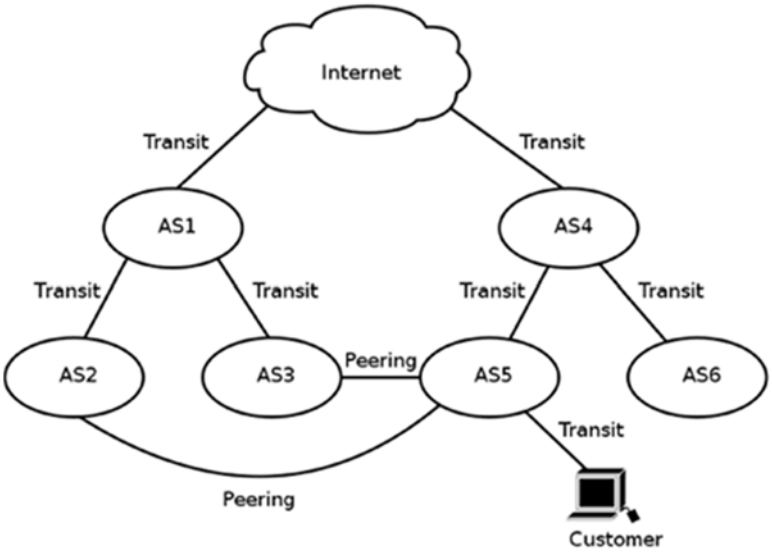

Figure 1 Peering and IP Transit in BGP Sessions.

Peering: Peering is a business practice where two or more autonomous networking systems directly connect with each other to exchange traffic, without the need for settlement. They can avoid complex traffic record-keeping and save costs, achieving a completely reciprocal commercial behaviour.

Transit: In this mode, one backbone network pays another backbone network for interconnectivity. It is a typical “provider-customer” business relationship where the customer

pays for transit fees to purchase services and gain access to other networks.

When the two networks are exchanging data (peering), it is only necessary to correctly transmit the routing information and downlink routes of the local network to the peer network through BGP to ensure the correct forwarding of data packets. This is relatively simple for peer-to-peer mode. In the IP Transit mode, it is necessary to provide lines of different qualities according to customer needs.

2.2 S-BGP Security Mechanism

The BGP has been widely used for over 30 years. However, with the rapid expansion of network scale, increasing openness and complexity of network environments, as well as the lack of security authentication mechanisms, the vulnerability of the BGP protocol has been fully exposed (Goldberg, 2014). Deliberate attacks by hackers and administrator misconfigurations can seriously hinder the safe and stable operation of the internet. Kent (2003) of the US BBN Technologies, received joint funding from the US National Security Agency (NSA) and the Defense Advanced Research Projects Agency (DARPA) and launched the first BGP security solution, S-BGP. This mechanism provides a comprehensive security solution for the BGP protocol from the IP layer to the application layer, including three primary security mechanisms: Public Key Infrastructure (PKI), confirmation attributes, and Internet Protocol Security (IPSec).

Although S-BGP resolved the security issues of BGP, it also introduced the following problems(Guo, 2005):

1) increased network overhead;
2) increased router computations;
3) increased network storage overhead;
4) difficult deployment; and
5) inability to resist joint attack.

In particular, the fourth point is significant. Firstly, there are substantial issues with the deployment of PKI. Communication resources are a strategic resource for a country, so a country cannot rely on a PKI managed by another country to ensure the security of its national communication network. Additionally, different operators in the same country have significant competition, which can also result in problems with PKI deployment. Second, new BGP routers

need to obtain their certificates and private keys first, which makes it difficult to expand the network. Therefore, as the most comprehensive security solution, S-BGP has not been favoured by the industrial sector.

2.3 The RPKI Security

Based on the research background and related work mentioned above, the Secure Inter-Domain Routing (SIDR) working group was established by IETF in 2006. In 2010, the RPKI technical standard was initiated based on the S-BGP mechanism as the technical prototype. A series of standard documents have been released since 2012, and the standardization of RPKI technology framework and related core protocols was declared completed by 2017(Chung et al., 2019). In the same year, IETF established the Secure Inter-Domain Routing Operations (SIDROPS) working group to promote RPKI and resolve issues encountered during deployment (Mitseva, Panchenko & Engel, 2018). With the support of various international and regional organizations related to the internet, such as the five Regional Internet Registries (RIR) and the National Institute f Standards and Technology (NIST) in the United States, RPKI has been rapidly promoted.

RPKI adopts the idea of S-BGP, which binds IP address prefixes and ASNs through signatures. However, in order to avoid modifying BGP and minimize the overhead of border routers, RPKI does not attach signatures to BGP announcements. Instead, the signatures are stored independently in a distributed RPKI repository and verified by dedicated servers. The router then uses the validated result to guide route selection. The compatibility with BGP and the out-of-band design philosophy are important reasons for the deployment of RPKI. IETF has been standardizing RPKI since 2012 (Hlavacek et al.2022). and continuously updating and improving concepts and standards related to RPKI. RPKI prevents prefix hijacking and sub-prefix hijacking by providing a trusted mapping between IP address prefixes and ASNs. It changes the “default accept” mode of BGP announcements to “default reject” mode, and only the route announcements judged as valid by RPKI will be accepted by the AS.

2.4 Development Status of RPKI in BGP

In the industry, major global vendors have released router hardware devices that support RPKI route origin validation functionality in order to gain a competitive advantage in new technologies and for their own market promotion. According to Zou Hui et al. (2022), Huawei has released NE05E, NetEngine8000, Me40 series, Cisco has the ISO/XR series, and Nokia

has the SROS series. Software developers or enthusiasts have also released some software tools that support RPKI route origin validation functionality. Representative examples include:

GoBGP: GoBGP is an open-source implementation of the BGP protocol, developed by Nippon Telegraph and Telephone (NTT)' Software Innovation Center, the largest telecommunications operator in Japan. It can be integrated with GUN Zebm and other tools to support multiple routing protocols(BGP implemented in the Go Programming Language, 2023).

BIRD: BIRD is an open-source implementation of a fully-featured routing daemon that supports multiple routing protocols, including BGP, Routing Information Protocol (RIP), Open Shortest Path First (OSPF), and route origin validation. BIRD originated from a student project at the Faculty of Mathematics and Physics at Charles University in Prague, Czech Republic. It has been applied in London and Moscow exchange centres (The BIRD Internet Routing Daemon, 2020).

Other tools that support RPKI route origin validation functionality include OpenBGPD, FRRouting, etc.

In academia, Zou Hui et al. (2022)analyzed the problem of unilateral revocation under the current RPKI architecture and the resulting risk of resource failure of subordinate Certificate Authorities (Cas). They proposed to improve the transparency of CA issuance behaviour by deploying a log server to record CA issuance behaviour and designed an efficient real-time monitoring and emergency response mechanism based on this. Bao Zhuo et al. (2022) proposed an RPKI-Autonomous System Provider Authorization(ASPA) improved BGP path protection mechanism. Xiao Wenlong et al. (2022) proposed an RPKI cache update conflict detection mechanism based on factual ownership. The mechanism uses the reverse RPKI to Router(RTR) protocol and RPKI data hierarchical distribution architecture to collect and synchronize factual routing origin information and detects conflicting RPKI cache update data by comparing factual routing origin information with RPKI cache update data, which protects the authenticity and validity of RPKI cache. Dr. He Guobiao (2021) from Beijing Jiaotong University proposed a decentralized trusted domain name root service mechanism based on blockchain to address many security risks of the internet's important infrastructure, such as excessive central authority, single-point failure, and data privacy leaks. Shrishak et al. (2020) proposed using threshold signature algorithms to require the consensus of the five major RIRs

for the issuance and revocation of each RPKI certificate, thereby limiting the absolute power of a single RIR. Xing Q et al. (2018) and Garcia-Martinez A et al. (2020) each proposed a decentralized solution based on blockchain, which utilizes the common characteristics of IP addresses, AS numbers, and digital currencies – transferability, allocatability, and inability to be simultaneously allocated to multiple entities – to achieve decentralization, traceability, and tamper-proof authentication of the allocation relationship information of Internet numbering resources.

In summary, the industry's technology and academic research results provide theoretical support and lay a practical foundation for the large-scale deployment of RPKI in BGP.

3. Methodology

The core of RPKI lies in the authentication of ownership and usage rights of IP address prefixes and AS numbers. RPKI includes two major components: a Certificate Authority (CA) system and a Relying Parties (RPs): the CA is responsible for issuing RPKI resource certificates and ROA (routing origin authentication) signatures; the RPs periodically download these certificates and signatures from the RPKI database synchronously. The router obtains these data from the RPs and provides them to the router. The router uses the RTR Protocol to judge the authenticity of the BGP routing message, thereby realizing the security check of the routing information. The project implements the security mechanism of RPKI according to this principle, using bird2 of Charles University in Prague, Czech Republic, as an Internet Routing Daemon for network routing protection.

3.1 Realization of RPKI mechanism

The basic idea of RPKI is to construct a PKI to authenticate ownership and usage rights of IP address prefixes and AS numbers. RPKI consists of two major components, the Certificate Authority (CA) system and the Relying Parties (RPs). The CA system is illustrated in Figure 2.

Each organization that registers and holds internet addresses and AS resources can act as a CA for RPKI, which is the entity that issues certificates. The CA has two main functions:

1. To authenticate the allocation of number resources, RPKI resource certificates are issued. When a CA allocates a portion of its IP address resources or AS numbers to a subordinate organization, it issues a resource certificate to confirm the allocation. The certificate contains the binding relationship between the IP address prefix/AS number and the

receiving organization.

2. To authorize an autonomous network to make a route announcement for a specific IP address prefix, a Route Origin Authorization (ROA) signature is issued. The ROA binds the AS number of the autonomous network with the IP prefix. These resource certificates and ROA signatures are stored and published in the repository maintained by each CA, and the distributed database composed of all these publication points is called the RPKI repository.

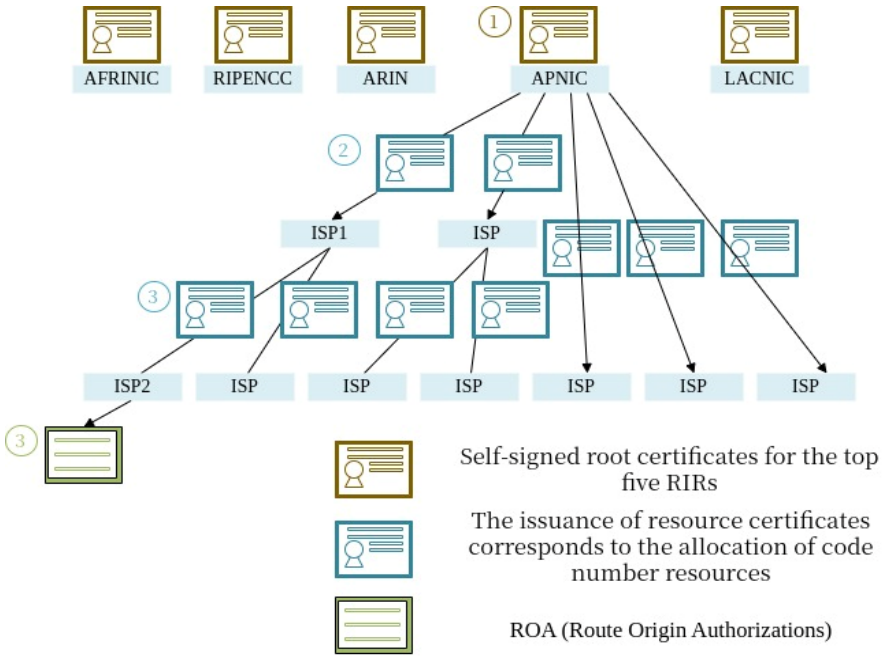

Figure 2: RPKI Resource Certificate Hierarchy

The RPKI Dependent Party (RP) is responsible for periodically synchronizing and downloading certificates and signatures from the RPKI repository, and verifying their validity to obtain the true authorization relationship between IP prefixes and AS numbers. Routers obtain this data from the RPKI dependent party to determine the authenticity of BGP route messages, that is, whether the originating AS in the route message has a legitimate authorization to announce the IP prefix. This is shown in Figure 3.

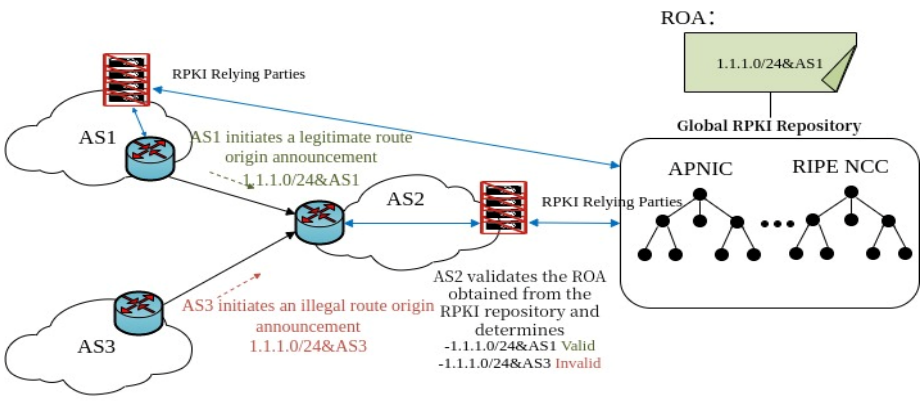

Figure 3: The RPKI Components

RPKI Relying Parties serve as a bridge between the RPKI system and the Internet inter-domain routing system, playing a crucial role in the IP address and routing security fields of the Internet's security authentication infrastructure. Each inter-domain routing participant, such as backbone network operators and ISPs, can deploy their own RP or choose to trust third-party RPs to handle RPKI information on their behalf. With the increasing popularity of cloud computing, these third-party RP services may gradually move to the cloud, making the positioning of RPKI services as a public service of the Internet more clear.

3.2 Realization of The RTR Protocol

The RTR protocol is used to synchronize data between validators and routers, including Classless Inter-Domain Routing (CIDR), ASN, MaxLength, etc. So that routers can verify the legitimacy of the routes. Due to frequent encryption and decryption on routers, a lot of CPU and memory resources are consumed, which can degrade router performance. RTR is a lightweight protocol that occupies very little memory. Therefore, routers do not need to perform frequent and complex encryption and validation algorithms.

In RTR, Protocol Data Units (PDUs) obtain caches from the cache server, and the steps for establishing and reconnecting connections are as figure 4.

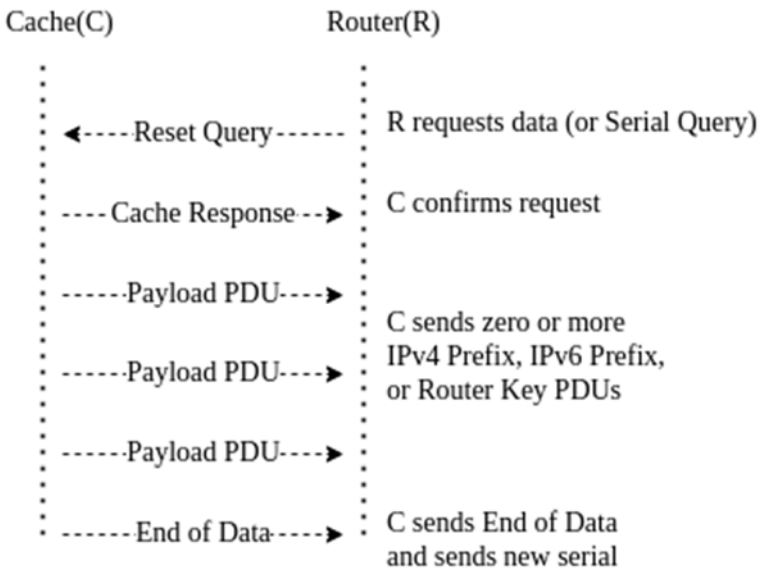

Figure 4: RTR Protocol Sequences: Start or Restart

The Ipv4 Prefix and Ipv6 Prefix in the Payload PDU are critical for updating the information from the cache server to the local device. The data packet contains information in the format of {Prefix, Len, Max-Len, ASN}. The main difference between Ipv6 Prefix and Ipv4 Prefix is that Ipv6 Prefix has an additional 96 bits due to the format of Prefix, Len, and Max-Len being specific to Ipv6.

4. Results

The AS built by the author is named AS47158, and it has three nodes, namely node1, node2, and node3. The AS47158 will provide IP transit service to customer AS142634, who requests to use BGP large communities to identify the origin of received routes from each node and use BGP large communities to control the router from advertising routes to the selected nodes.

As shown in Figure 5, AS47158 peers with AS1335 in node 3. AS59947 provides IP transit service to AS47158, but customers are not allowed to use AS59947's routes. AS47158 also peers with AS1335 in node2, and AS6939 provides IP transit service to AS47158 in node2. Additionally, AS142634 is allowed to use this route.

Node2 and node3 are connected to node1 through iBGP, and node1 provides IP transit service to AS142634.

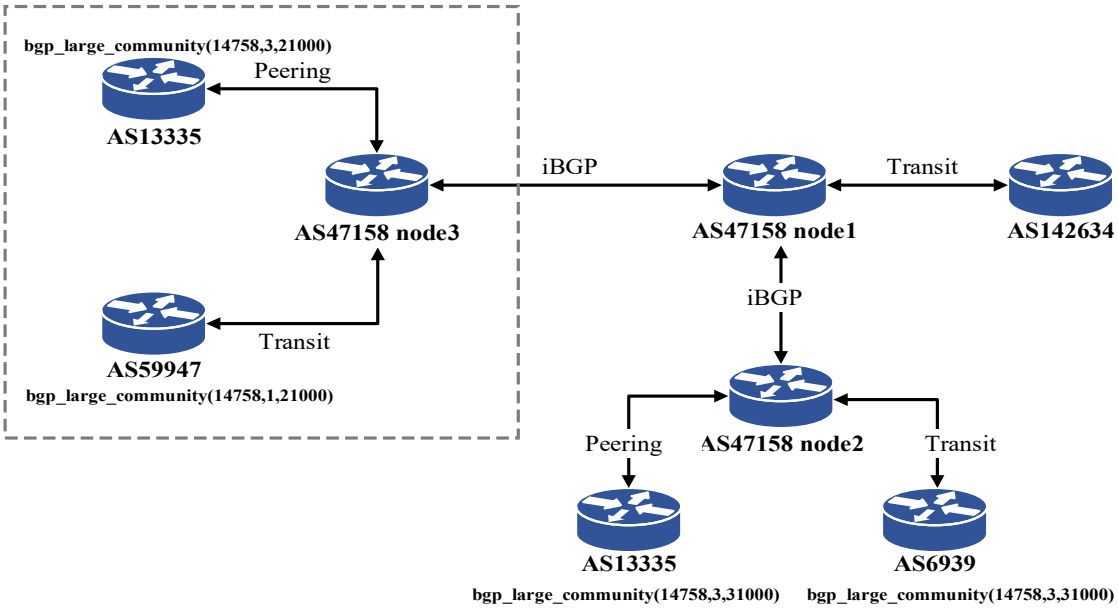

Figure 5: Network Topology Diagram

4.1 The architecture of the router system

The router system framework is shown in Figure 6 (the dashed part of Figure 5 network topology). The BGP configuration name "bgp_own" is used for importing or exporting routes that are only available to the local network. All traffic from upstream will be matched by the source MAC address and marked with fwmark 0xea2b. All traffic with fwmark 0xea2b will be routed using Table 247. Outbound traffic does not need to be matched because inbound traffic matching is performed when connecting to internal nodes, and the routing table is already selected for inbound traffic, from which outbound traffic will read its routing.

The BGP configuration name "bgp" is similar to "bgp_own", but it is for downstream preparation. Table 256 is a mixed routing table of Table 147 and Table 247, which can be used when two upstream mixed lines are required.

In the internal nodes of AS47158, we use iBGP for connection. The iBGP configuration name "ibgp" is used to transmit routes that are only available to AS47158 (the local network). We use a virtual line with VLAN ID 5 to transmit packets from AS47158, that is, adding VLAN ID to packets from the local network, and using "from all iif ens20.5 lookup 247" to select the routing table on both sides.

The iBGP configuration name "ibgp" is used to transmit routes for downstream use. The method is similar, but we do not use VLAN ID to reduce transmission overhead, although the additional overhead caused by VLAN ID is not significant. Then, we use "from all iif ens20

lookup 247" to select the routing table.

Each node has a unique BGP large community identification tag so that subsequent customers can adjust the priority from each node or control the route announcement to a specific node according to their needs. The advantage of using VLAN ID is that no additional physical lines are required, and when additional separate lines are needed, creating another virtual line using VLAN is sufficient.

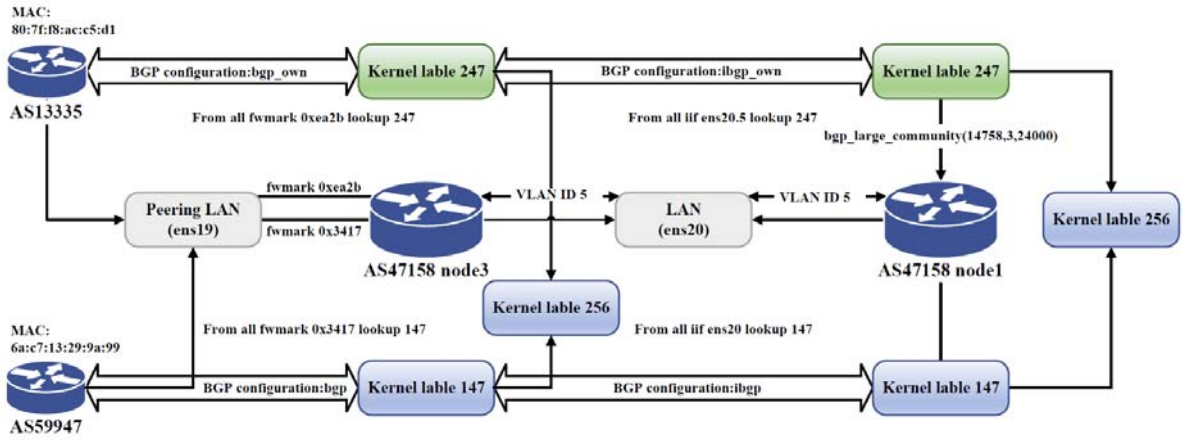

Figure 6: Software Framework Diagram

4.2 Software Implementation

The author utilized bird2 as an Internet Routing Daemon for network routing protection. Bird2 is a student project developed by the Faculty of Mathematics and Physics at Charles University in Prague, Czech Republic. It is characterized by its programmability and implementation of multiple dynamic routing protocols (such as Babel, BGP, and OSPF), and is an open-source software under the GNU General Public License.

BGPQ4 is a software that can retrieve data from an IRR database, and we will use it to obtain prefixes from the AS-SET for IRR filtering. We used Debian11 (on nodes 2 and 3) and Arch Linux (on node 2) as the operating systems, although any Linux distribution could be used.

4.2.1 Import Routing Table

Routing security is an important consideration when configuring BGP, and we need to prevent BGP hijacking. We will perform IRR verification, RPKI verification, next_hop verification, and source ASN verification for users, upstream and peer partners.

For importing filters from customers(The codes are in Appendix 5), as shown in Figure 7, Node 1 will perform AS_path checking on the routes from customers in the first step to prevent AS_path spoofing. The second step will check if there is a Tier1 ISP ASN in the routes from customers, and if there is, the route will be rejected. The third step checks if the next hop is the peer's next hop to prevent traffic redirection to other networks. The fourth step checks if the customer's routes have corresponding Route6 records in IRR and whether RPKI is INVALID. This prevents customer configuration errors from hijacking addresses of other networks. The fourth step deletes BGP large communities used for internal systems to prevent BGP large communities from being maliciously exploited. The fifth step adds BGP large communities for identifying the source of routes, and the sixth step adds BGP large communities for identifying downstream routes that need to be exported. Finally, the filtered routes are imported in the seventh step.

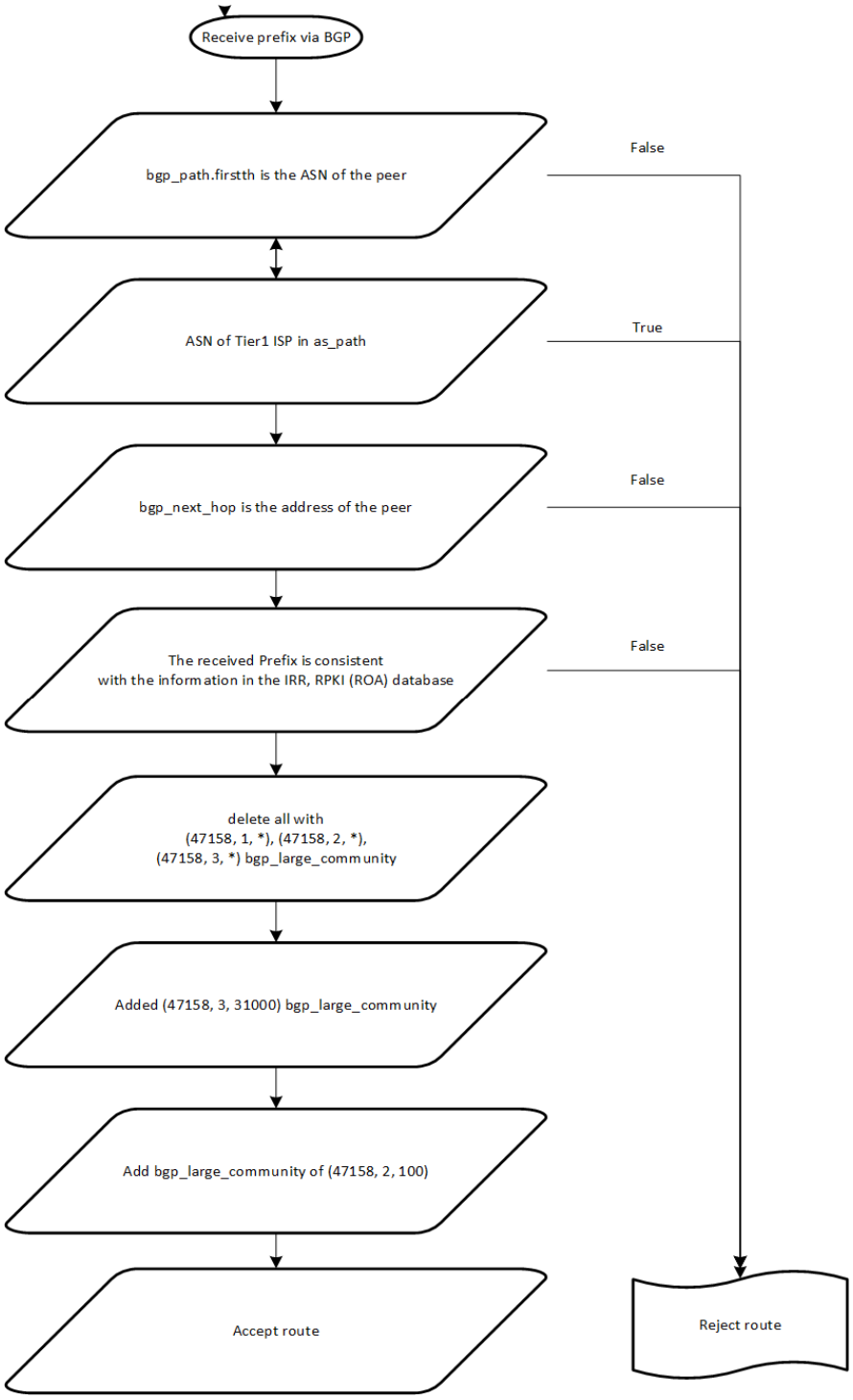

Figure 7 Process Flowchart for Prefix Import from Customer

For importing filters from our upstream AS6939 or peer networks AS13335 and AS59947 (upstream but not allowed for customer use), there are differences in the import filters. When importing with AS13335, AS6939, and AS59947 sessions, the (14758, 2, 100) BGP large communities (indicating that the routes come from downstream and need to be exported) will not be added. Additionally, the BGP session will not filter Tier 1 ISP ASN, meaning that there

is no second step as shown in Figure 7. The BGP large communities added by AS13335 and AS59947 in the last step are also different. As shown in Figure 8(The codes are in Appendix 7), the BGP large communities added by AS13335 in the penultimate step indicate that they can be used by downstream, while those added by AS59947 indicate that they cannot be used by downstream.

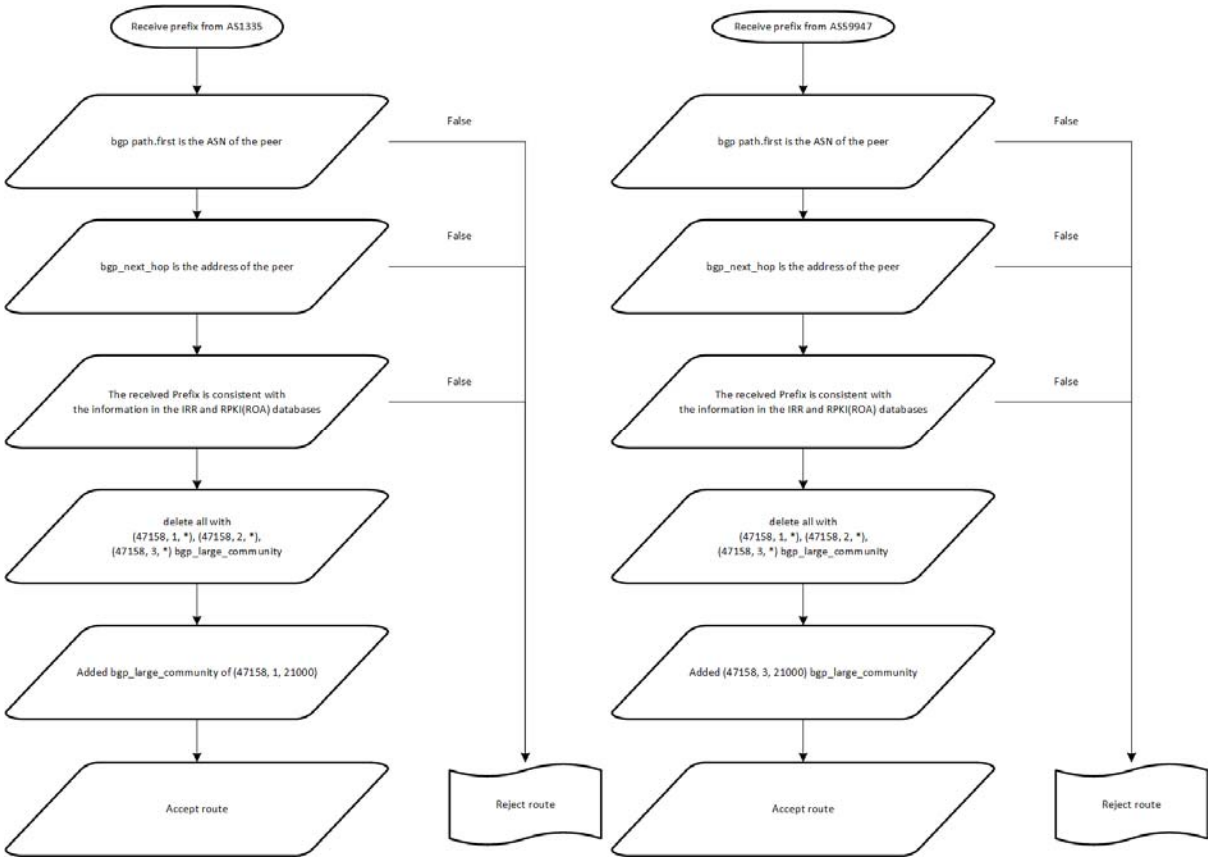

Figure 8 Filtering Process for AS13335 and AS59947

4.2.2 Export Route

The author created three functions at all nodes, namely bgp_export(), which is used to export routes from the local network, bgp_export_downstream(), which exports routes sent by downstream, and bgp_export_all(), which is used for route transmission between different nodes in the local network. These three functions only need to be modified for the node ID (bgp_large_community) at each of the three nodes. For example, the bgp large community for target 2 is (14758, 4, 21000) at node 3, and only needs to be changed to (14758, 4, 23000) at node 1. The code for these functions is as follows:

```
#used to export the route of this network
```

```
function bgp_export()
{
    my_opt_prefix(); #Perform IRR verification on the address to be declared on this website
    if is_bogon() then return false; #Filter BOGON prefix
    if bgp_large_community ~ [(14758, 4, 23000)] then return false;#goal 2
    if proto = "BGP_Prefix" then return true;#Prefixes that need to be announcelocally
    if source != RTS_BGP then return false;#Reject routes not from BGP
    if bgp_large_community !~ [(47158, 2, 1)] then return false;#Reject routes that do not declare labels    return true;
}
#Used between iBGP
function bgp_export_all() {
    if bgp_export() then return true;
    if is_bogon_asn() then return false;
    if is_bogon_prefix() then return false;
    if source != RTS_BGP then return false;
    return true;
}
#Used to export routes from downstream
function bgp_export_downstream() {
    if is_bogon_asn() then return false;
    if is_bogon_prefix() then return false;
    if source != RTS_BGP then return false;
    if bgp_large_community ~ [(47158, 4, 23000)] then return false;#goal 2
    if bgp_large_community !~ [(47158, 2, 100)] then return false;#Export routes from downstream
    return true;
}
```

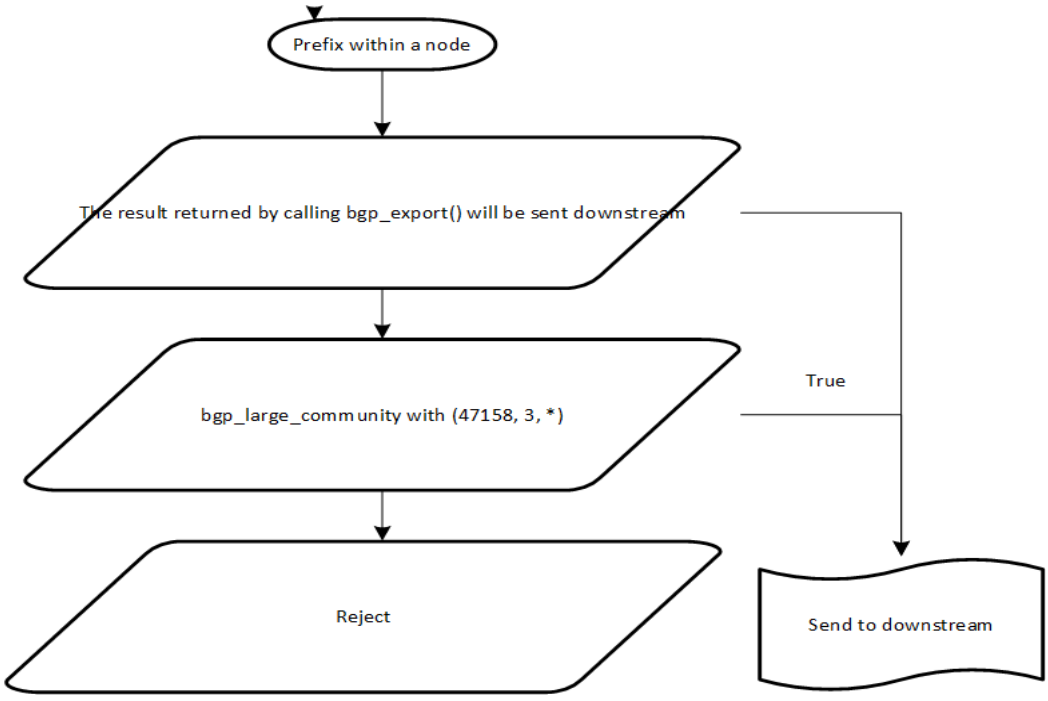

Figure 9 For the Prefix Export (Export) Flowchart for Customers

To route traffic to downstream AS142634, it is necessary to export the routes from AS47158 (local network), AS13335 (peer network), and AS6939 (upstream network), as shown in Figure 9(The code to establish a BGP connection with AS6939 is in Appendix 6). To achieve this, the return results of both bgp_export() and bgp_export_downstream() should be exported for AS13335 and AS6939. However, for AS59947, only the routes from AS47158 (local network) need to be sent, so bgp_export() should be used separately.

4.2.3 Synchronous in the Kernel and Between Various Route Tables

The bird filter can only select one routing table within the same session and cannot read from both bgp_v6_own and bgp_v6 simultaneously. Therefore, we need to synchronize the routes that need to be announced, as shown in Figure 10. When we perform policy routing(The specific implementation is in appendix 1 and appendix 2), all traffic from the peer will be routed (data forwarding) using the selected routing table (bgp_v6_own or bgp_v6), which will also include TCP packets from BGP. If there is no route to the peer in the routing table, the BGP connection will be disconnected. To avoid errors in importing or exporting routes and causing the node to lose network connection, making it impossible to remotely control the node, we create a new routing table, default6 (Table 256), instead of directly importing it into Linux's main routing table (main).

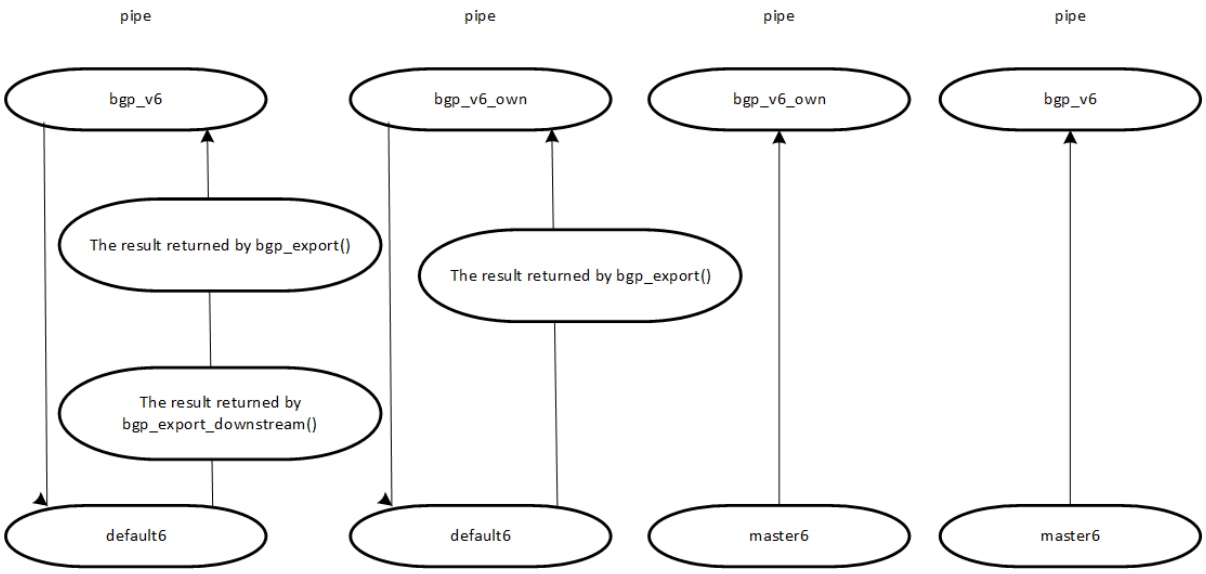

Figure 10: Illustration of Synchronization between Routing Tables

We import the main routing table from Linux to Bird in order to work with the device and direct protocols in Bird and raise the priority of local routes. This prevents the peer's address from appearing in the routes sent by BGP during BGP sessions, which can cause a loop and

result in BGP session interruption. We need to use the kernel protocol in Bird on all nodes to achieve the import/export of routing within the Linux kernel as illustrated in Figure 10 and Figure 11 (The codes are shown in Appendices 3).

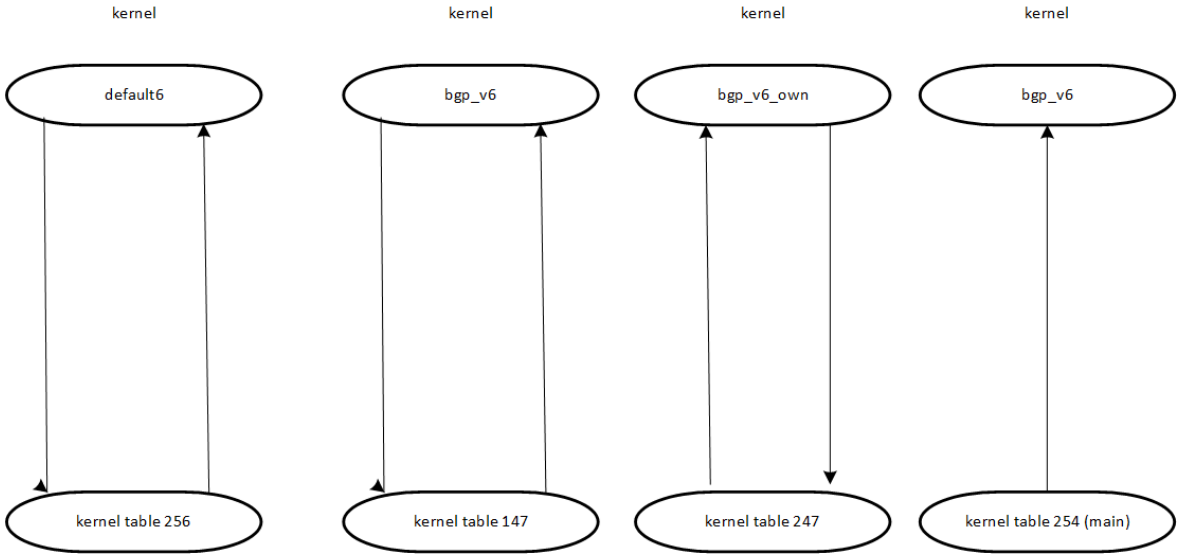

Figure 11: Illustration of Importing or Exporting Routes into the Kernel

5. Discussion

RPKI prevents source prefix hijacking and source sub-prefix hijacking by providing a trusted mapping between IP address prefixes and ASNs. It changes the "accept by default" mode of BGP announcements to the "deny by default" mode, and only the route announcement will be accepted by the AS. Thus, the secure routing mechanism set by the RPKI mechanism is realized, but any security mechanism limited by various factors is not perfect, and there are still various challenges.

5.1 The Role and Challenges of RPKI

The RPKI only verifies if the origin ASN and prefix, as well as the length of the announced prefix, match. Attackers can bypass this validation mechanism by modifying the origin ASN. Therefore, “enforce first AS” should be enabled when establishing a BGP session to prevent neighbours from modifying the first ASN. However, this does not completely solve the problem as hijackers can still bypass RPKI by modifying the AS-PATH after the first ASN, just not by modifying the first ASN in the AS-PATH.

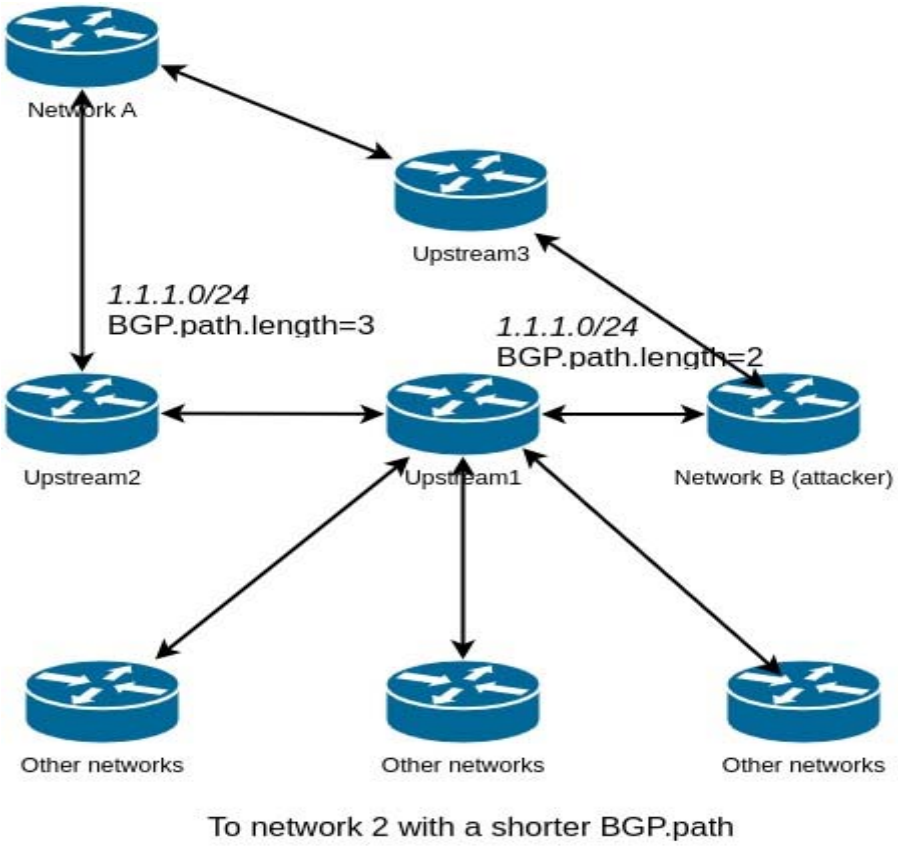

Figure 12: For Upstream 1 and Downstream of Upstream 1, to Network 2 has Higher Priority.

Here are the specific methods of hijacking:

Assume that Network A hosts a banking website and that compared to other networks, the AS-PATH between the attacker (Network B) and Network A is shorter. As shown in Figure 12, Upstream1 has a shorter AS-PATH to Network 2, with an AS-PATH length of only 1 to Network B. However, reaching Network A requires passing through Upstream2 (which has an AS-PATH length of 2). This means that all downstream "other network" traffic from Upstream1 will go to Network B, the attacker.

To further carry out the attack, Network B can add Network A's ASN to its own AS-SET, causing filters on other networks to add Network B's prefixes. In this way, Network B can then announce Network A's prefix to the other networks.

In response, an attacker could deploy a reverse proxy to the source site (the bank's website) on Network A. and use HTTP authentication to request a TLS/SSL certificate as a way to perform a man-in-the-middle attack. If the bank will monitor the issuance of TLS/SSL certificates, the hijacker can use HTTP directly, which will reduce the likelihood of detection.

Figure 13: RPKI Can Potentially be Circumvented Through the Modification of the AS_PATH.

Assuming that the ASN of the hijacker is AS2, the ASN of the victim is AS1, and the prefix targeted for hijacking is 1.1.1.0/24. By modifying the AS_PATH, the attacker can bypass RPKI and hijack the route with AS1 being the source ASN. Figure 13 illustrates the difference in AS_PATH and whether it passes RPKI verification.

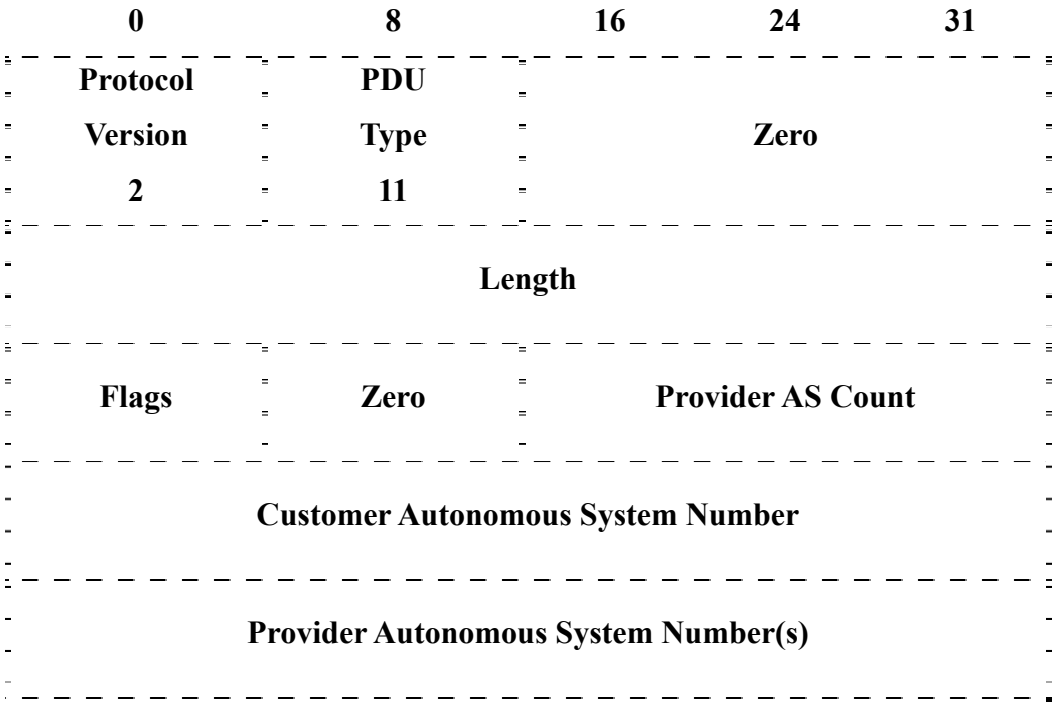

Figure 14: The RTR protocol with ASPA.

To tackle this issue, Alexander Azimov et al(2020). submitted a draft to the IETF named ASPA (Autonomous System Provider Authorization). ASPA includes an additional field called "Provider AS Count," depicted in Figure 14, which enables the effective validation of the upstream Autonomous System Provider (AS) and prevents BGP hijacking by "carrying downstream."

5.2 Low Deployment Rate

Despite the benefits that RPKI provides in terms of securing BGP, its adoption rate remains relatively low. According to data from the National Institute of Standards and Technology (NIST), as of February 2023, the deployment rate of RPKI in global IPv6 routing stands at a mere 45.48%(shown in figure 15). When it comes to IPv4, the situation is even less encouraging, with an RPKI deployment rate of only 42.4%(shown in figure 16).

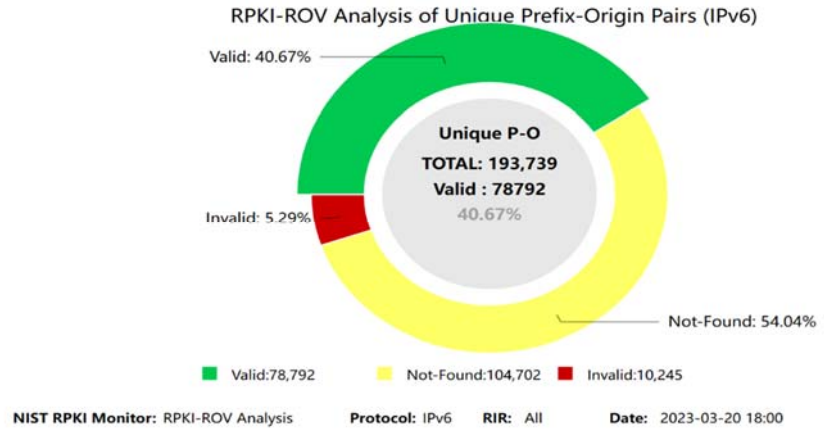

Figure 15: IPV6 Schematic diagram of layout ratio

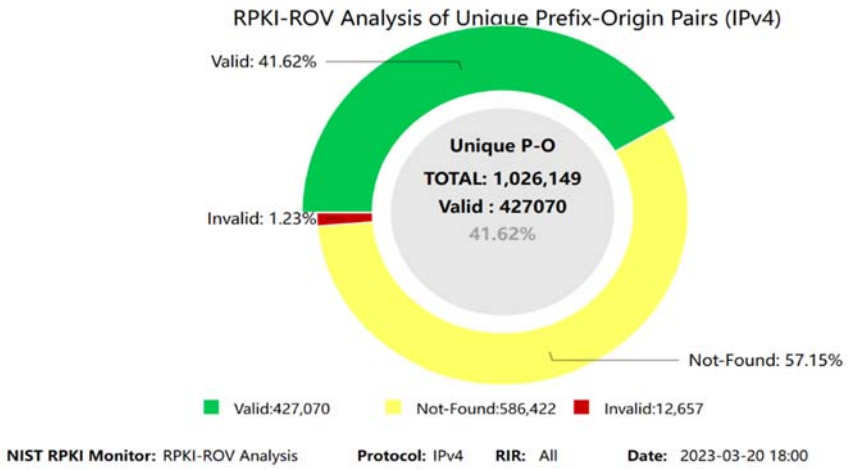

Figure 16: IPV4 Schematic diagram of layout ratio

Several factors may have contributed to this low deployment rate.

First, RPKI implementation is complex and requires significant technical expertise. Network administrators and engineers need to invest time and effort in understanding the intricacies of RPKI and how to properly deploy it in their networks.

Second, the deployment of RPKI involves additional costs, including those in infrastructure upgrades and ongoing maintenance. For smaller Internet Service Providers (ISPs) and organizations, these costs can be prohibitive .

Lastly, there may be a lack of awareness about the significance of RPKI and its potential benefits among network operators. This can result in a lack of motivation to adopt the technology, especially if other security measures are already in place.

To address the issue of low RPKI deployment, it is essential to promote awareness of its benefits and facilitate the adoption process. This can be done by providing more accessible resources, training, and support for network operators, as well as promoting incentives for RPKI deployment. By improving the adoption rate of RPKI, we can strengthen the security of BGP routing and mitigate the risks associated with hijacking attacks.

6. Conclusion

The project utilizes VLAN and Mac-based policy routing to fully utilize network resources and provide IP Transit and BGP Peering to customers on-demand without the need for additional devices. It also addresses packet matching and route announcement control using bgp large communities, as well as source identification on the same interface. In addition, our network project (Autonomous System numbers AS 47158 and AS 142634, website: https://huize.asia https://ixp.su) has established BGP sessions for IP Transit with Hurricane Electric and a peering relationship with Cloudflare. The solution has been deployed in our Poland, Italy, United States, and Taiwan nodes, providing IP Transit services for seven Autonomous Networks. However, the functionality of bgp large communities is still limited, and customers cannot adjust route black-holing or priorities using bgp large communities. Moreover, compared to other intelligent routing platforms, we do not have a visualization panel, load balancing, automated deployment, or coordination with hardware, which can be improved in future work. In terms of security, the ASPA is still in draft form, and it can effectively prevent BGP hijacking through downstream carry while bypassing RPKI. Currently, bgp.tools provides

an automatic alert function when an ASN is added to an AS-SET, which effectively provides a warning and detects the possible occurrence of hijacking for network operators to prevent it. For the low deployment rate of RPKI, the Mutually Agreed Norms for Routing Security (MANRS) has initiated a project that requires its members to deploy RPKI, which will effectively increase the deployment rate of RPKI.

References

1) Azimov, A., Bogomazov, E., Bush, R., Patel, K. and Snijders, J., 2020. Verification of AS_PATH Using the Resource Certificate Public Key Infrastructure and Autonomous System Provider Authorization. *Network Working Group Internet Draft. November*.
2) BAO Zhuo, MA Di, MAO Wei, et al.2022. Improved BGP Path Protection Mechanism Based on RPKI-ASPA. *Computer Systems & Applications*, 31(2), pp.316-324.
3) Chung, T., Aben, E., Bruijnzeels, T., Chandrasekaran, B., Choffnes, D., Levin, D., Maggs, B.M., Mislove, A., Rijswijk-Deij, R.V., Rula, J. and Sullivan, N., 2019, October. RPKI is coming of age: A longitudinal study of RPKI deployment and invalid route origins. In *Proceedings of the Internet Measurement Conference* (pp. 406-419).
4) García-Martínez, A., Angieri, S., Liu, B., Yang, F. and Bagnulo, M., 2020. Design and implementation of inblock—a distributed ip address registration system. *IEEE Systems Journal*, *15*(3), pp.3528-3539.
5) GitHub - osrg/gobgp: BGP implemented in the Go Programming Language https://github.com/osrg/gobgp
6) Goldberg, S., 2014. Why is it taking so long to secure internet routing?. *Communications of the ACM*, *57*(10), pp.56-63.
7) Griffin, T.G. and Wilfong, G., 1999. An analysis of BGP convergence properties. ACM SIGCOMM Computer Communication Review, 29(4), pp.277-288.
8) Guo Li. 2005. Research on security mechanism of S-BGP protocol. *Shanxi: Xidian university*, 2005
9) HE Guo-Biao. 2021. Research on Key Technologies of Decentralized Trustworthy Internet Infrastructure. *Beijing: Beijing Jiaotong University*.

10) Hlavacek, T., Jeitner, P., Mirdita, D., Shulman, H. and Waidner, M., 2022, November. Behind the scenes of RPKI. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1413-1426).

11) Kent, S., Lynn, C. and Seo, K., 2000. Secure border gateway protocol (S-BGP). *IEEE Journal on Selected areas in Communications*, *18*(4), pp.582-592.

12) Kent, S.T., 2003, October. Securing the border gateway protocol: A status update. In *Communications and Multimedia Security* (Vol. 2828, pp. 40-53.

13) Lepinski, M. and Kent, S., 2012. *An infrastructure to support secure internet routing* (No. rfc6480).

14) Mitseva, A., Panchenko, A. and Engel, T., 2018. The state of affairs in BGP security: A survey of attacks and defenses. *Computer Communications*, *124*, pp.45-60.

15) Oorschot, P.V., Wan, T. and Kranakis, E., 2007. On interdomain routing security and pretty secure BGP (psBGP). *ACM Transactions on Information and System Security (TISSEC)*, *10*(3), pp.11-es.

16) Rekhter, Y., Li, T. and Hares, S., 2006. *A border gateway protocol 4 (BGP-4)* (No. rfc4271).

17) Shrishak, K. and Shulman, H., 2020, July. Limiting the power of RPKI authorities. In *Proceedings of the Applied Networking Research Workshop* (pp. 12-18).

18) The BIRD Internet Routing Daemon Project (network.cz) https://bird.network.cz/

19) White, R., 2003. Securing BGP through secure origin BGP (soBGP). *Business Communications Review*, *33*(5), pp.47-53.

20) Wählisch, M., Maennel, O. and Schmidt, T.C., 2012. Towards detecting BGP route hijacking using the RPKI. *ACM SIGCOMM Computer Communication Review*, *42*(4), pp.103-104..

21) XIAO Wen-Long, MA Di, MAO Wei, et al. 2022. Fact Ownership-based Conflict Detection Scheme for RPKI Cache Update. *Computer Systems & Applications*, 31(2), pp.366-375.

22) Xing, Q., Wang, B. and Wang, X., 2018. Bgpcoin: Blockchain-based internet number resource authority and bgp security solution. *Symmetry*, *10*(9), p.408.

23) ZOU Hui, LI Yanbiao, YU Chenhui, MA Di, et al.2022. A Revocation Detection Mechanism in RPKI Based on Behavior Transparency[J]. *Frontiers of Data&Computing*, 4(3), pp.90-109.

24) Zou Hui, MA Di, SHAO Qing, MAO Wei, et al.2022. A Survey of the Resource Public Key Infrastructure. *Chinese Journal of Computers,* (5):45.

Appendix 1: Policy Routing of Mac Addresses in Node 3

In Node 3, we discovered through packet capturing that the MAC addresses of AS59947 and AS13335 are 6a:c7:13:29:9a:99 and 80:7f:f8:ac:c5:d1, respectively.

Therefore, we used nftables to match the MAC addresses and labeled them with a fwmark. Then, we selected the routing table through ip rule. The implementation is as follows. We can directly enter decimal numbers here because ip rule and nftables automatically convert the input to hexadecimal.

```
table ip6 mangle {
        chain PREROUTING {
                type filter hook prerouting priority mangle; policy accept;
                iif "eth1" ether saddr 80:7f:f8:ac:c5:d1 meta mark set 13335
                iif "eth1" ether saddr 6a:c7:13:29:9a:99 meta mark set 0x000
0ea2b
        }
}
```

```
ip -6 rule add from all fwmark 0x0000ea2b lookup 247
ip -6 rule add from all fwmark 13335 table 147
```

Appendix 2: Adding VLAN in Linux

To add a VLAN with VLAN ID 5 on ens19

```
ip link add link ens19 name ens19.5 type vlan id 5
ip link set ens19.5 up
ip a a fe80::5247/64 dev ens19.5
```

Appendix 3: Internal iBGP and Kernel Route Synchronization

```
protocol kernel {
    kernel table 256;
    learn;
        ipv6 {
                table default6;# Create a default route table.
                import all;
                export filter {
                        #krt_prefsrc = 2602:feda:ab2::1;
                #krt_prefsrc = 2a0c:e640:101a::1;
```

```
                accept;
            };
        };
}

protocol kernel {               # For bgp that allow downstream
        kernel table 147;
        ipv6 {
                table bgp_v6;
                export filter {
                    accept;
                };
                import all;
        };
}

protocol kernel {               # For bgp that don't allow downstream
        kernel table 247;
        ipv6 {
                table bgp_v6_own;
                export all;
                import all;
        };
}

protocol kernel {
        ipv6 {          #To import the main route in Linux into BIRD
                export filter {
                    reject;
                };
                import all;
        };
}



protocol device {
        scan time 60;
}

protocol direct {
    ipv6 {
    import all;
    export all;
    };
}

```

```
58.
59.
60. #1. To import the routing entries from the main routing table in Linux to the
    BGP_v6 table (table 147)
61. protocol pipe {
62.         table master6;
63.         peer table bgp_v6;
64.         export all;
65.         import none;
66. }
67. #2. To import the routes from the main routing table in Linux to the
    bgp_v6_own table (table 247):
68. protocol pipe {
69.         table master6;
70.         peer table bgp_v6_own;
71.         export all;
72.         import none;
73. }
74. #3. To implement routing announcement that appears in two routing tables,
75. protocol pipe {                          # Synchronize the routing table that
    can carry downstream and cannot be written to the main routing table
76.         table bgp_v6;
77.         peer table default6;
78.         export all;
79.         import filter {
80.             if bgp_export() then accept;
81.             if bgp_export_downstream() then accept;
82.             reject;
83.         } ;                    #Synchronize the prefixes that the local
    machine needs
```

The tpl_ibgp template is for iBGP (using route reflectors) and is used to propagate routes that downstream devices are allowed to use. On the other hand, the tpl_ibgp_own template is used to propagate routes that are restricted for use within the local network only (not allowed for downstream devices).。

```
1.  template bgp tpl_ibgp {
2.      local as LOCAL_ASN;
3.      graceful restart;
4.      rr client;
5.      direct;
6.     ipv6 {
7.         table bgp_v6;
8.         next hop self;
9.         import filter {
10.             if is_bogon_asn() then {
11.                 print "is bogon asn", net, " for ASN ", bgp_path.last;
```

```
                reject;
        }
            if is_bogon_prefix() then {
                print "is bogon prefix", net, " for ASN ", bgp_path.last;
                reject;
            }
            accept;
        };
        export filter {
            if proto = "BGP_Prefix" then bgp_large_community.add((47158, 2,
1));# To add BGP large community to identify which addresses need to be
announced within the local network
            if bgp_export_all() then accept;
            reject;
        };
    };
}


template bgp tpl_ibgp_own {
    # ibgp transfer table which cannot have downstream
    local as LOCAL_ASN;
    graceful restart;
    rr client;
    direct;
   ipv6 {
       table bgp_v6_own;
       next hop self;
       import filter {
            if is_bogon_asn() then {
                print "is bogon asn", net, " for ASN ", bgp_path.last;
                reject;
        }
            if is_bogon_prefix() then {
                print "is bogon prefix", net, " for ASN ", bgp_path.last;
                reject;
            }
            accept;
        };
        export filter {
            if proto = "BGP_Prefix" then bgp_large_community.add((47158, 2,
1));#同上
            if bgp_export_all() then accept;
            reject;
        };
    };
}
```

Appendix 4: Timely Retrieval of Information from IRR Database

Create a script file at /etc/bird/bgpq4_prefix.sh and include the following content.。

```
as_set=("AS-HUIZE" "AS47158")
prefix=$(bgpq4 -b -A -6 ${as_set[*]} -R 48 -l allowed_prefix)
if [[ "$prefix" =~ ^ERROR.* ]]; then
    exit
else
echo "$prefix" > /etc/bird/filters/allowed_prefix.conf
fi

as_set=("as-ixpsu AS-WUT-EXCHANGE")
prefix=$(bgpq4 -b -A -6 ${as_set[*]} -R 48 -l AS142634)
if [[ "$prefix" =~ ^ERROR.* ]]; then
    exit
else
echo "$prefix" > /etc/bird/filters/AS142634.conf
fi

/usr/sbin/birdc 'c'
```

```
crontab -e

*/5 * * * * timeout 10s bash /etc/bird/bgpq4_prefix.sh >/dev/null #实现定时刷新
```

Appendix 5: Configuration file for BGP sessions with downstream devices on Node 1

```
function is_AS142634()
prefix set AS142634;
int set AS142634_origins;
{
    include "/etc/bird/filters/AS142634.conf"; # Internet Routing Registries Filter Function
    if net !~ AS142634 then {
        print " net",net," is not from AS142634";
        return false;
    }
    return true;
}


protocol bgp customer_as142634 from tpl_bgp {
    source address 2406:840:e240:1:1410:11::;
```

```
16.     neighbor 2406:840:e240:1::3 as 142634;
17.     ipv6 {
18.         table bgp_v6;
19.         next hop self;
20.         import filter{
21.
22.             # Prevent BGP PATH Hijacking
23.             if (bgp_path.first != 142634 ) then {
24.                 print "FILTERED_FIRST_AS_NOT_PEER_AS: ", net, " ", bgp_path;
25.                 reject;
26.             }
27.             reject_transit_paths();
28.
29.             # Prevent BGP NEXT_HOP Hijacking
30.             if !( from = bgp_next_hop ) then {
31.                     print "FILTERED_NEXT_HOP_NOT_PEER_IP: ", net, " ",
    bgp_path,"next_hop:"," ",bgp_next_hop ;
32.                     reject;
33.             }
34.
35.             # RPKI validation
36.             if (roa_check(r6, net, bgp_path.last) = ROA_INVALID) then
37.             {
38.                     print "Ignore RPKI invalid ", net, " for ASN ",
    bgp_path.last;
39.                    reject;
40.             }
41.             # IRR validation
42.
43.             is_AS142634();
44.             bgp_large_community.delete([(47158, 2, *)]);
45.             bgp_large_community.delete([(47158, 1, *)]);
46.             bgp_large_community.delete([(47158, 3, *)]);
47.             bgp_large_community.add((47158, 3, 23000));
48.             bgp_large_community.add((47158, 2, 100));
49.             accept;
50.         };
51.
52.         export filter{
53.             if bgp_export() then accept;
54.             if bgp_large_community ~ [(47158, 3, *)] then accept;
55.             reject;
56.         };
57.     };
58. }
```

Appendix 6: BGP Session with AS6939

```
function is_AS_HURRICANER()
prefix set AS_HURRICANE;
int set AS_HURRICANE_origins;
{
    include "/etc/bird/filters/AS-HURRICANE.conf";
    if net !~ AS_HURRICANE then {
        return false;
    }
    return true;
}


protocol bgp AS6939 from tpl_bgp {
    source address 2a0f:5707:ffe3::57;
    neighbor 2a0f:5707:ffe3::30 as 6939;
    ipv6 {
        table bgp_v6;
        preference 115;
        next hop self;
        import filter{
            if is_bogon_asn() then {
                    print "is bogon asn", net, " for ASN ", bgp_path.last;
                    reject;
            }
            if is_bogon_prefix() then {
                    print "is bogon prefix", net, " for ASN ", bgp_path.last
;
                    reject;
            }
            if (roa_check(r6, net, bgp_path.last) = ROA_INVALID) then
            {
                    print "Ignore RPKI invalid ", net, " for ASN ", bgp_path
.last;
                   reject;
            }
            is_AS_HURRICANER();
            bgp_large_community.delete([(47158, 2, *)]);
            bgp_large_community.delete([(47158, 1, *)]);
            bgp_large_community.delete([(47158, 3, *)]);
            bgp_large_community.add((47158, 3, 31000));
            accept;
        };

        export filter{
            if bgp_export_downstream() then accept;
            else reject;
```

```
        };
    };
}
```

Appendix 7: Code on Node 3

The code on other nodes is similar but may require some modifications. However, due to the length of the code, it will not be displayed here.、

```
/etc/bird/bird.conf


define LOCAL_ASN = 47158;

router id 5.101.92.125;

roa4 table r4;
roa6 table r6;
ipv6 table default6;
ipv6 table bgp_v6;
ipv6 table bgp_v6_own;
ipv4 table default4;
ipv4 table bgp_v4;  #For allowed to carry downstream
ipv4 table bgp_v4_own; #For not allowed to carry downstream


log syslog all;

protocol static BGP_Prefix{
    ipv6 { table default6; };
    route 2602:feda:ab2::/48 reject;
    route 2a06:a005:5ab::/48 reject;
}

protocol static BGP_Prefix_v4{
    ipv4  { table default4; };
    route 179.61.216.0/24 reject;
}

protocol kernel {
    kernel table 256;
    learn;
#    persist;
        ipv6 {
                table default6;
```

```
                import all;
                export filter {
          #krt_prefsrc = 2602:feda:ab2::1;
                #krt_prefsrc = 2a0c:e640:101a::1;
                accept;
            };
        };
}

protocol kernel {
        kernel table 256;
        ipv4 {
                table default4;
                import all;
                export filter {
                #krt_prefsrc = 5.189.255.107;
                accept;
            };
        };
}

protocol kernel {               # For bgp that allow downstream
        kernel table 147;
        ipv6 {
                table bgp_v6;
                export filter {
                    accept;
                };
                import all;
        };
}

protocol kernel {               # For bgp that allow downstream
        kernel table 147;
        ipv4 {
                table bgp_v4;
                export filter {
                    accept;
                };
                import all;
        };
}

protocol kernel {               # For bgp that don't allow downstr
   eam
        kernel table 247;
        ipv4 {
                table bgp_v4_own;
                export filter {
                    accept;
                };
```

```
                import all;
        };
}


protocol kernel {               # For bgp that don't allow downstream
        kernel table 247;
        ipv6 {
                table bgp_v6_own;
                export all;
                import all;
        };
      }

      protocol kernel {
              ipv6 {
                      export filter {
                          reject;
                      };
                      import all;
              };
      }

      protocol kernel {
              ipv4 {
                      export filter {
                          reject;
                      };
                      import all;
              };
      }

      protocol device {
       scan time 60;
      }

      protocol direct {
              #interface "dummy*";
          ipv6 {
          import all;
          export all;
          };
      }

      function net_len_too_long(){
          case net.type {
              NET_IP4: return net.len > 24; # IPv4 CIDR greater than /24 is too long
```

```
135.              NET_IP6: return net.len > 48; # IPv6 CIDR greater
   than /48 is too
   long        #else: print "net_len_too_long: unexpected net.type ",
    net.type, " ", net; return false;
136.          }
137.      }
138.
139.      define BOGON_ASNS = [
140.          0,                      # RFC 7607
141.          23456,                  # RFC 4893 AS_TRANS
142.          64496..64511,           # RFC 5398 and documentation/ex
   ample ASNs
143.          64512..65534,           # RFC 6996 Private ASNs
144.          65535,                  # RFC 7300 Last 16 bit ASN
145.          65536..65551,           # RFC 5398 and documentation/ex
   ample ASNs
146.          65552..131071,          # RFC IANA reserved ASNs
147.          4200000000..4294967294, # RFC 6996 Private ASNs
148.          4294967295              # RFC 7300 Last 32 bit ASN
149.      ];
150.      define BOGON_PREFIXES_V4 = [
151.          0.0.0.0/8+,             # RFC 1122 'this' network
152.          10.0.0.0/8+,            # RFC 1918 private space
153.          100.64.0.0/10+,         # RFC 6598 Carrier grade nat sp
   ace
154.          127.0.0.0/8+,           # RFC 1122 localhost
155.          169.254.0.0/16+,        # RFC 3927 link local
156.          172.16.0.0/12+,         # RFC 1918 private space
157.          192.0.2.0/24+,          # RFC 5737 TEST-NET-1
158.          192.88.99.0/24+,        # RFC 7526 deprecated 6to4 rela
   y anycast. If you wish to allow this, change `24+` to `24{25,32}`(
   no more specific)
159.          192.168.0.0/16+,        # RFC 1918 private space
160.          198.18.0.0/15+,         # RFC 2544 benchmarking
161.          198.51.100.0/24+,       # RFC 5737 TEST-NET-2
162.          203.0.113.0/24+,        # RFC 5737 TEST-NET-3
163.          224.0.0.0/4+,           # multicast
164.          240.0.0.0/4+            # reserved
165.      ];
166.      define BOGON_PREFIXES_V6 = [
167.          ::/8+,                  # RFC 4291 IPv4-
   compatible, loopback, et al
168.          0100::/64+,             # RFC 6666 Discard-Only
169.          2001::/32{33,128},      # RFC 4380 Teredo, no more spec
   ific
170.          2001:2::/48+,           # RFC 5180 BMWG
171.          2001:10::/28+,          # RFC 4843 ORCHID
172.          2001:db8::/32+,         # RFC 3849 documentation
173.          2002::/16+,             # RFC 7526 deprecated 6to4 rela
   y anycast. If you wish to allow this, change `16+` to `16{17,128}`
   (no more specific)
```

```
174.          3ffe::/16+,              # RFC 3701 old 6bone
175.          fc00::/7+,               # RFC 4193 unique local unicast
176.          fe80::/10+,              # RFC 4291 link local unicast
177.          fec0::/10+,              # RFC 3879 old site local unica
   st
178.          ff00::/8+                # RFC 4291 multicast
179.      ];
180.
181.      function is_bogon_prefix() {
182.          case net.type {
183.              NET_IP4: return net ~ BOGON_PREFIXES_V4;
184.              NET_IP6: return net ~ BOGON_PREFIXES_V6;
185.              #else: print "is_bogon_prefix: unexpected net.type
   ", net.type, " ", net; return false;
186.          }
187.      }
188.
189.      function is_bogon_asn() {
190.          if bgp_path ~ BOGON_ASNS then return true;
191.          return false;
192.      }
193.
194.      protocol rpki {
195.      #        debug all;
196.
197.              roa4 { table r4; };
198.              roa6 { table r6; };
199.
200.              # Please, do not use rpki-
   validator.realmv6.org in production
201.              remote "rtr.rpki.cloudflare.com" port 8282;
202.
203.              retry keep 5;
204.              refresh keep 30;
205.              expire 600;
206.      }
207.
208.      filter peer_in_v6 {
209.              if is_bogon_asn() then {
210.                      #print "is bogon asn", net, " for ASN ", bg
   p_path.last;
211.                      reject;
212.              }
213.              if is_bogon_prefix() then {
214.                      #print "is bogon prefix", net, " for ASN ",
    bgp_path.last;
215.                      reject;
216.              }
217.              if (roa_check(r6, net, bgp_path.last) = ROA_INVALID
   ) then
218.              {
```

```
219.                     #print "Ignore RPKI invalid ", net, " for A
   SN ", bgp_path.last;
220.                     reject;
221.             }
222.             accept;
223.     }
224.
225.
226.     function is_bogon() {
227.         if is_bogon_asn() then return true;
228.         if is_bogon_prefix() then return true;
229.         if net_len_too_long() then return true;
230.         return false;
231.     }
232.
233.     function my_opt_prefix()
234.     prefix set allowed_prefix;
235.     int set allowed_prefix_origins;
236.     {
237.         include "/etc/bird/filters/allowed_prefix.conf";
238.             case net.type {
239.             NET_IP4: return true;
240.             NET_IP6: if net ~ allowed_prefix then return true;
241.             else: print "net_is_not_from_the_as_set: unexpected
    net.type ", net.type, " ", net; return false;
242.         }
243.     }
244.
245.     function bgp_export()
246.     {
247.         my_opt_prefix();
248.         if is_bogon() then return false;
249.         if bgp_large_community ~ [(47158, 4, 21000)] then retur
   n false;
250.         if proto = "BGP_Prefix" then return true;
251.         if source != RTS_BGP then return false;
252.         if bgp_large_community !~ [(47158, 2, 1)] then return f
   alse;
253.         return true;
254.     }
255.
256.     function bgp_export_all() {
257.         if bgp_export() then return true;
258.         if is_bogon_asn() then return false;
259.         if is_bogon_prefix() then return false;
260.         if source != RTS_BGP then return false;
261.         return true;
262.     }
263.
264.     function bgp_export_downstream() {
265.         if is_bogon_asn() then return false;
```

```
266.          if is_bogon_prefix() then return false;
267.          if source != RTS_BGP then return false;
268.          if bgp_large_community ~ [(47158, 4, 21000)] then retur
   n false;
269.          if bgp_large_community !~ [(47158, 2, 100)] then return
    false;
270.          return true;
271.      }
272.
273.      protocol pipe {                          # Synchronize the r
   outing table that can carry downstream and cannot be written to th
   e main routing table
274.              table bgp_v6;
275.              peer table default6;
276.              export all;
277.              import filter {
278.                  if bgp_export() then accept;
279.                  if bgp_export_downstream() then accept;
280.                  reject;
281.              } ;                     #Synchronize the prefixes th
   at the local machine needs to announce into 2 tables for BGP
282.      }
283.
284.      protocol pipe {
285.              table bgp_v6_own;
286.              peer table default6;
287.              export all;
288.              import filter {
289.                  if bgp_export() then accept;
290.                  reject;
291.              } ;
292.      }
293.
294.      protocol pipe {                          # Synchronize the r
   outing table that can carry downstream and cannot be written to th
   e main routing table
295.              table bgp_v4;
296.              peer table default4;
297.              export all;
298.              import filter {
299.                  if bgp_export() then accept;
300.                  if bgp_export_downstream() then accept;
301.                  reject;
302.              } ;                     #Synchronize the prefixes th
   at the local machine needs to announce into 2 tables for BGP
303.      }
304.
305.      protocol pipe {
306.              table bgp_v4_own;
307.              peer table default4;
308.              export all;
```

```
309.              import filter {
310.                  if bgp_export() then accept;
311.                  reject;
312.              } ;
313.      }
314.
315.      protocol pipe {
316.              table master6;
317.              peer table bgp_v6;
318.              export all;
319.              import none;
320.      }
321.
322.      protocol pipe {
323.              table master6;
324.              peer table bgp_v6_own;
325.              export all;
326.              import none;
327.      }
328.
329.      protocol pipe {
330.              table master4;
331.              peer table bgp_v4;
332.              export all;
333.              import none;
334.      }
335.
336.      protocol pipe {
337.              table master4;
338.              peer table bgp_v4_own;
339.              export all;
340.              import none;
341.      }
342.
343.      template bgp tpl_bgp_v4 {
344.          allow bgp_local_pref on;
345.          graceful restart on;
346.          local as LOCAL_ASN;
347.          ipv4 {
348.              table bgp_v4;
349.              preference 110;
350.              next hop self;
351.              import filter{
352.                  if is_bogon_asn() then {
353.                          print "is bogon asn", net, " for ASN ",
    bgp_path.last;
354.                          reject;
355.                  }
356.                  if is_bogon_prefix() then {
357.                          print "is bogon prefix", net, " for ASN
    ", bgp_path.last;
```

```
                            reject;
                    }
                    if (roa_check(r6, net, bgp_path.last) = ROA_INV
   ALID) then
                    {
                            print "Ignore RPKI invalid ", net, " fo
   r ASN ", bgp_path.last;
                           reject;
                    }
                    bgp_large_community.delete([(47158, 2, *)]);
                    bgp_large_community.delete([(47158, 1, *)]);
                    bgp_large_community.delete([(47158, 3, *)]);
                    bgp_large_community.add((47158, 3, 21000));
                    accept;
                };

                export filter{
                    if bgp_export() then accept;
                    if bgp_export_downstream() then accept;

                    reject;
                };
            };
        }

        template bgp tpl_bgp_v4_own {
            allow bgp_local_pref on;
            graceful restart on;
            local as LOCAL_ASN;
            ipv4 {
                table bgp_v4_own;
                preference 110;
                next hop self;
                import filter{
                    if is_bogon_asn() then {
                            print "is bogon asn", net, " for ASN ",
    bgp_path.last;
                            reject;
                    }
                    if is_bogon_prefix() then {
                            print "is bogon prefix", net, " for ASN
    ", bgp_path.last;
                            reject;
                    }
                    if (roa_check(r6, net, bgp_path.last) = ROA_INV
   ALID) then
                    {
                            print "Ignore RPKI invalid ", net, " fo
   r ASN ", bgp_path.last;
                           reject;
                    }
```

```
403.                    bgp_large_community.delete([(47158, 2, *)]);
404.                    bgp_large_community.delete([(47158, 1, *)]);
405.                    bgp_large_community.delete([(47158, 3, *)]);
406.                    bgp_large_community.add((47158, 1, 21000));
407.                    accept;
408.                };
409.
410.                export filter{
411.                    reject;
412.                };
413.            };
414.        }
415.
416.        template bgp tpl_bgp {
417.            allow bgp_local_pref on;
418.            graceful restart on;
419.            local as LOCAL_ASN;
420.            ipv6 {
421.                table bgp_v6;
422.                preference 110;
423.                next hop self;
424.                import filter{
425.                    if is_bogon_asn() then {
426.                            print "is bogon asn", net, " for ASN ",
    bgp_path.last;
427.                            reject;
428.                    }
429.                    if is_bogon_prefix() then {
430.                            print "is bogon prefix", net, " for ASN
    ", bgp_path.last;
431.                            reject;
432.                    }
433.                    if (roa_check(r6, net, bgp_path.last) = ROA_INV
   ALID) then
434.                    {
435.                            print "Ignore RPKI invalid ", net, " fo
   r ASN ", bgp_path.last;
436.                           reject;
437.                    }
438.                    if net ~ 2001:468:d01::/48 then if proto = "Moe
   IX" then preference = 115;
439.                    bgp_large_community.delete([(47158, 2, *)]);
440.                    bgp_large_community.delete([(47158, 1, *)]);
441.                    bgp_large_community.delete([(47158, 3, *)]);
442.                    bgp_large_community.add((47158, 3, 21000));
443.                    accept;
444.                };
445.
446.                export filter{
447.                    if bgp_export() then accept;
448.                    if bgp_export_downstream() then accept;
```

```
449.                 else reject;
450.             };
451.         };
452.     }
453.
454.     template bgp tpl_bgp_own {
455.         allow bgp_local_pref on;
456.         graceful restart on;
457.         local as LOCAL_ASN;
458.         ipv6 {
459.             table bgp_v6_own;
460.             preference 110;
461.             next hop self;
462.             import filter{
463.                 if is_bogon_asn() then {
464.                         print "is bogon asn", net, " for ASN ",
    bgp_path.last;
465.                         reject;
466.                 }
467.                 if is_bogon_prefix() then {
468.                         print "is bogon prefix", net, " for ASN
    ", bgp_path.last;
469.                         reject;
470.                 }
471.                 if (roa_check(r6, net, bgp_path.last) = ROA_INV
   ALID) then
472.                 {
473.                         print "Ignore RPKI invalid ", net, " fo
   r ASN ", bgp_path.last;
474.                        reject;
475.                 }
476.                 bgp_large_community.delete([(47158, 2, *)]);
477.                 bgp_large_community.delete([(47158, 1, *)]);
478.                 bgp_large_community.delete([(47158, 3, *)]);
479.                 bgp_large_community.add((47158, 3, 21000));
480.                 accept;
481.             };
482.
483.             export filter{
484.                 if bgp_export() then accept;
485.                 else reject;
486.             };
487.         };
488.     }
489.
490.     template bgp tpl_ibgp {
491.         local as LOCAL_ASN;
492.         graceful restart;
493.         rr client;
494.         direct;
495.        ipv6 {
```

```
496.             table bgp_v6;
497.             next hop self;
498.             import filter {
499.                 if is_bogon_asn() then {
500.                     print "is bogon asn", net, " for ASN ", bgp
    _path.last;
501.                     reject;
502.              }
503.                 if is_bogon_prefix() then {
504.                     print "is bogon prefix", net, " for ASN ",
    bgp_path.last;
505.                     reject;
506.                 }
507.                 accept;
508.              };
509.              export filter {
510.                 if proto = "BGP_Prefix" then bgp_large_communit
    y.add((47158, 2, 1));
511.                 if bgp_export_all() then accept;
512.                 reject;
513.              };
514.          };
515.         ipv4 {
516.             table bgp_v4;
517.             next hop self;
518.             import filter {
519.                 if is_bogon_asn() then {
520.                     print "is bogon asn", net, " for ASN ", bgp
    _path.last;
521.                     reject;
522.              }
523.                 if is_bogon_prefix() then {
524.                     print "is bogon prefix", net, " for ASN ",
    bgp_path.last;
525.                     reject;
526.                 }
527.                 accept;
528.              };
529.              export filter {
530.                 if proto = "BGP_Prefix_v4" then bgp_large_commu
    nity.add((47158, 2, 1));
531.                 if bgp_export_all() then accept;
532.                 reject;
533.              };
534.          };
535.      }
536.
537.
538.      template bgp tpl_ibgp_own {
539.          # ibgp transfer table which cannot have downstream
540.          local as LOCAL_ASN;
```

```
541.          graceful restart;
542.          rr client;
543.          direct;
544.         ipv6 {
545.             table bgp_v6_own;
546.             next hop self;
547.             import filter {
548.                 if is_bogon_asn() then {
549.                     print "is bogon asn", net, " for ASN ", bgp
    _path.last;
550.                     reject;
551.              }
552.                 if is_bogon_prefix() then {
553.                     print "is bogon prefix", net, " for ASN ",
    bgp_path.last;
554.                     reject;
555.                 }
556.                 if net ~ 2001:468:d01::/48 then if proto = "Moe
    IX" then preference = 115;
557.                 accept;
558.              };
559.              export filter {
560.                 if proto = "BGP_Prefix" then bgp_large_communit
    y.add((47158, 2, 1));
561.                 if bgp_export_all() then accept;
562.                 reject;
563.              };
564.          };
565.      }
566.
567.      include "/etc/bird/peers/*";
568.
569.      protocol bgp AS13335 from tpl_bgp {
570.          source address 2001:678:4fc::92:125;
571.          neighbor 2001:678:4fc::92:163 as 13335;
572.      }
573.
574.
575.      protocol bgp AS13335_IPV4 from tpl_bgp_v4{
576.          source address 5.101.92.125;
577.          neighbor 5.101.92.163 as 13335;
578.      }
579.
580.      protocol bgp LL_IX_RS1_IPV6 from tpl_bgp_own {
581.          neighbor 2001:678:4fc::9%eth1 as 59947;
582.      }
583.
584.      protocol bgp LL_IX_RS2_IPV6 from tpl_bgp_own {
585.          neighbor 2001:678:4fc::2%eth1 as 59947;
586.      }
587.
```

```
588.       protocol bgp LL_IX_RS1_IPV4 from tpl_bgp_v4_own {
589.           source address 5.101.92.125;
590.           neighbor 5.101.92.9 as 59947;
591.       }
592.
593.
594.       protocol bgp LL_IX_RS2_IPV4 from tpl_bgp_v4_own {
595.           source address 5.101.92.125;
596.           neighbor 5.101.92.2%eth1 as 59947;
597.       }
598.
599.       protocol bgp node1 from tpl_ibgp {
600.           neighbor fe80::5247%'ens20' as 47158;
601.       }
602.
603.       protocol bgp node1_own from tpl_ibgp_own {
604.           neighbor fe80:1::5247%'ens20.5' as 47158;
605.       }
```

```
1. /etc/bird/add_rules.bash
2.
3. #!/bin/bash
4. ip -6 rule add from all iif ens20.5 table 247
5. ip -6 rule add from all iif ens20 table 147
6. ip -6 rule add from all fwmark 59947 table 247
7. ip rule add from all fwmark 59947 lookup 247
8. ip -6 rule add from all fwmark 13335 table 147
9. ip  rule add from all fwmark 13335 table 147
```

```
1. /etc/bird/bgpq4_prefix.sh
2.
3. #!/bin/bash
4. as_set=("AS-HUIZE" "AS141011")
5. prefix=$(bgpq4 -b -A -6 ${as_set[*]} -R 48 -l allowed_prefix)
6. if [[ "$prefix" =~ ^ERROR.* ]]; then
7.     exit
8. else
9. echo "$prefix" > /etc/bird/filters/allowed_prefix.conf
10.fi
11.
12./usr/sbin/birdc 'c'
13.
```