University of Amsterdam

Research Project 2

Comparative Analysis of ASPA Implementations

July 7, 2026

Student:
Benedict Rigler, Nils Claessens
15658821,13708449

Supervisor:
Luuk Hendriks
(luuk@nlnetlabs.nl)

Abstract

The Border Gateway Protocol (BGP) was not designed with security in mind, and is thus fundamentally vulnerable to route hijacking and route leaks. While an existing solution, RPKI-ROV, has seen significant adoption, it is limited to verifying route origin and cannot detect illegitimate intermediate hops or leaked paths. Autonomous System Provider Authorization (ASPA) is designed to bridge this gap by allowing ASes to cryptographically sign their authorised upstream providers, enabling valley-free path validation.

However, the transition from IETF draft to global deployment presents several critical challenges. This project investigates whether different BGP implementations (BIRD and OpenBGPD) reach the same conclusion when validating identical AS_PATHs, how they handle complex topological edge cases, and how performance compares between daemons, and compared to running them without ASPA.

This project utilises a containerised test harness using Containerlab and Docker, incorporating Routinator as the Relying Party and ExaBGP as a scriptable BGP injector.

Testing across $\pm 762K$ prefixes found consistent ASPA validation behaviour between both daemons, suggesting the specification yields reliable, interoperable results. However, validation mode configuration was found to have a significant impact on routing decisions, with the daemons in upstream mode producing a 25% route rejection rate in the current setup. Performance overhead was found to be comparable to ROV, which is encouraging for adoption. Limitations include lack of available daemons, and low ASPA adoption rate ($\pm 2\%$) at the time of testing.

Contents

1 Introduction

The Border Gateway Protocol (BGP) is the "glue" that binds the internet together, enabling thousands of Autonomous Systems (ASes) to exchange information on what Internet Protocol (IP) addresses each knows how to reach. However, BGP was created in an era where the internet was still small, and the expectation of mutual trust was reasonable. This means that BGP was not designed with malicious actors in mind, leaving it inherently vulnerable to both malicious attacks and accidental configuration mistakes, the biggest of which are route hijacking and route leaks.

Route hijacking occurs when an AS advertises a prefix it is not authorised to originate, either to intercept traffic (man-in-the-middle), cause a denial of service, or redirect users to fraudulent infrastructure [1]. High-profile examples include the 2018 hijack of Amazon's Route 53 DNS prefixes to steal cryptocurrency [2] and the 2020 Rostelecom incident affecting over 8,000 prefixes belonging to Google, Cloudflare, and others [3]. Route leaks are typically accidental misconfigurations in which an AS propagates a route beyond its intended scope, inadvertently offering transit it should not provide. The 2019 Cloudflare outage-caused by a small Pennsylvania ISP leaking routes that Verizon then propagated globally-illustrates how a single misconfiguration can destabilise large portions of the internet [4].

While manual route filters can act as a mitigation, they are difficult to maintain at scale. The most prominent mitigation in use is Resource Public Key Infrastructure Route Origin Validation (RPKI-ROV) [5]. It has achieved meaningful global adoption by allowing operators to verify that the origin AS of a route is authorised to announce a given prefix via Route Origin Authorizations (ROAs). However, RPKI-ROV is fundamentally limited to origin validation: it cannot detect illegitimate intermediate hops or route leaks. Autonomous System Provider Authorization (ASPA) [6] is designed to close this gap by allowing each AS to cryptographically register its upstream providers in the RPKI, enabling valley-free path validation of the full AS_PATH, i.e., ensuring that AS paths respect the customer-provider and peer-peer hierarchy and do not contain illegitimate "valleys" where traffic would incorrectly traverse back down the hierarchy.

1.1 Problem Statement

Although the ASPA specification provides a clear mathematical model for path validation, the transition from an IETF draft to a robust, global deployment presents several critical challenges:

- **Implementation Consistency:** The ASPA path validation algorithm is complex, especially when handling complex relationships between neighbours. Do different BGP daemons (e.g., BIRD, OpenBGPD) interpret the draft identically, or will they reach different conclusions when validating the same path?
- **ASPA Behaviour:** ASPA is meant to support partial adoption. How will BGP daemons behave on the real internet, given the current state of adoption?
- **Performance Scalability:** Validating an AS_PATH requires multiple lookups in the RPKI database for every BGP update. What impact do these additional operations have on the processing time of incoming advertisements?

1.2 Research Questions

This thesis seeks to evaluate the operational readiness and technical robustness of the ASPA ecosystem. To achieve this, the following research questions are addressed:

RQ1 Do BIRD and OpenBGPD reach the same conclusion when validating identical AS_PATHs under various RPKI and ASPA adoption states?

RQ2 How do BIRD and OpenBGPD behave when presented with data from the wider internet, which currently only has partial ASPA adoption?

RQ3 How does the performance of BIRD and OpenBGPD get impacted by the addition of ASPA?

2 Ethical Considerations

As this research involves the simulation of core internet routing protocols and the fabrication of cryptographic security records, several ethical considerations have been made to ensure the integrity of the project, as well as that of the global internet.

The primary risk in BGP is accidental leakage of experimental routes. This problem is mitigated by our isolated setup, where BGP peerings are only established between the containers inside the lab.

The research involves testing the limits and finding potential bugs in the active path validation implementations. As such, any bugs and/or vulnerabilities will be reported to the relevant parties according to the OS3 Responsible Disclosure policy.

The data used in this project consists entirely of either fabricated or publicly-available data. No private traffic, data, or configurations are used, ensuring there are no privacy concerns.

3 Background

The global internet relies on the cooperative exchange of reachability information among thousands of independently-managed Autonomous Systems (ASes). This inter-domain topology is maintained via the Border Gateway Protocol (BGP), which operates as a path-vector protocol: advertising the entire path to reach a certain destination [7]. BGP routers establish explicit point-to-point sessions with adjacent networks to exchange these paths; in the context of BGP, these adjacent routers are formally referred to as **neighbours** (or peers). Routing decisions are made based on local policy, combined with the primary piece of information inside BGP: the AS_PATH attribute.

The AS_PATH lists all ASes an advertisement has traversed, where the originating source is the right-most AS in the path, and the immediate neighbour from which the advertisement was received is the left-most (not considering complicated setups like route servers and internet exchanges) [7]. Because BGP was designed in the context of a small-scale internet, where cooperation and trust were valid assumptions, it lacks built-in security [8]. For example, BGP has no way to verify the authenticity of received advertisements, making the protocol susceptible to misconfigurations and malicious actors.

3.1 Routing Anomalies

The inherent lack of cryptographic verification in BGP enables both malicious actors and accidental misconfigurations to propagate invalid routing information. These routing anomalies primarily undermine traffic integrity, data privacy, and global network stability. In practice, inter-domain routing failures manifest through two distinct operational mechanisms: **route hijacking**, where an AS illegitimately originates a prefix it does not own, and **route leaks**, where an AS improperly propagates valid prefixes in violation of agreed-upon local routing policies [9, 1]. While both anomalies exploit BGP's trust-based path selection logic, they differ fundamentally in their technical execution, intentionality, and propagation dynamics [9].

3.1.1 Route Hijacking

Route Hijacking occurs when an AS advertises a prefix that it is not authorised to originate [10].

These malicious advertisements can be doctored to make them more likely to be chosen during the BGP best-path selection process. This is typically achieved through two primary methods:

- **More-Specific Prefix Injection:** By announcing a longer prefix (e.g., a /24 when the legitimate owner announces a /22), the hijacker exploits the "longest prefix match" rule in IP routing [10].
- **AS-Path Shortening:** A hijacker may offer a (seemingly) shorter route to the destination by spoofing the AS_PATH, making the malicious route appear more attractive to BGP's distance-vector logic [10].

When one of these malicious routes is chosen and ends up in the routing table, this can cause traffic to:

- Go the wrong way, but still be delivered, allowing for man-in-the-middle (MITM) attacks where data is transparently intercepted and monitored [10, 8].
- Get black-holed, leading to a (partial) denial of service [10, 3].
- Be directed to the wrong destination, such as a fraudulent server designed to harvest credentials or spread malware [10, 2, 8].

These incidents occur with alarming frequency. In 2018, attackers hijacked Amazon's Route 53 DNS prefixes to redirect users of MyEtherWallet to a phishing site, resulting

in the theft of cryptocurrency [2]. More recently, in 2020, a significant routing incident involving Rostelecom (AS12389) saw over 8,000 prefixes hijacked, including those belonging to big companies like Google, Facebook, Akamai, Cloudflare and Amazon, highlighting the ongoing fragility of the global routing table [3].

3.1.2 Route Leaks

While hijacks are often deliberate attempts to subvert traffic, a similar level of disruption can occur through unintentional violations of routing policy, known as **Route Leaks**. It is important to note that an advertisement does not state whether it was made maliciously or on accident. Therefore, it is often only determinable after the fact whether it was one or the other.

A route leak occurs when routing advertisement(s) are propagated beyond their intended scope, resulting in the redirection of traffic through a path that violates intended routing policy [1]. Essentially, it occurs when an AS receives a route from one provider or peer and incorrectly advertises it to another provider or peer, inadvertently offering transit services it should not provide.

Unlike hijacks, route leaks are frequently the result of accidental misconfigurations in "prefix-lists" or "route-maps." This usually happens in a multi-homed customer environment [4]:

- **Policy Violation:** A customer AS C learns a route from Provider A. Instead of keeping that route internal, C mistakenly advertises it to Provider B.
- **Path Selection:** Because Provider B likely prefers a "customer-learned" route over a "peer-learned" or "provider-learned" route, the leaked path is chosen, and global traffic begins flowing through AS C [1, 11].

When a leak occurs, the consequences are often felt globally due to the sheer volume of traffic diverted:

- **Congestion and Brown-outs:** Small customer links are overwhelmed by transit traffic they were never designed to handle, leading to massive packet loss [1].
- **Increased Latency:** Traffic may take geographically suboptimal paths (e.g., traffic between two European cities being routed through a leak in Asia) [8].
- **Traffic Interception:** Similar to hijacks, leaked routes allow an intermediate AS to inspect or drop traffic that should never have crossed its network [8].

Route leaks also occur regularly, and can destabilise large parts of the internet. In 2019, a small ISP in Pennsylvania accidentally leaked routes from its provider to Verizon. Verizon accepted these routes and propagated them globally, causing a massive portion of the world's traffic destined for Cloudflare to be funnelled through a bottleneck in a small regional network [4]. Another major incident occurred in 2021, when Vodafone Italy inadvertently leaked more than 30,000 BGP prefixes to global transit providers, disrupting services for more than an hour [12, 13].

3.2 Existing Mitigations

Several security measures have been proposed, with varying degrees of adoption and success. The simplest defence mechanism is the use of **Route Filters**. These are manual or semi-automated lists maintained by network operators to define which prefixes a neighbour is permitted to announce. To scale this process, operators historically relied on the **Internet Routing Registry (IRR)**, a distributed database network where operators register their routing policies and prefix ownership using standard description formats [14]. Crucially, the IRR was originally intended to serve as a passive, advisory repository for internet-wide debugging, network planning, and structural modelling [14, 15]. However, as the rapid expansion of the internet made manual filter updates operationally impossible, operators began utilising the IRR as a programmatic source for automatic route filter generation [15, 16]. Software suites like the IRRToolSet allowed transit providers to parse public RPSL data and script automated router configuration files directly, minimising human configuration errors [15, 16].

While these database-driven filters helped automate configuration, they remain notoriously difficult to maintain accurately at scale. Empirical measurements show that the global IRR ecosystem suffers from heavy data decay: over 18% of long-standing routing objects in legacy and third-party databases are completely stale, reflecting unmaintained or abandoned topologies that no longer exist [17, 18]. Furthermore, the IRR cannot provide an accurate, comprehensive topology because many

network operators intentionally omit or sanitise their policy data. Since writing comprehensive RPSL rules explicitly exposes an AS's private peering agreements, financial footprints, and traffic engineering strategies, major transit providers deliberately restrict what they publish to protect proprietary business secrets from competitors [19]. Consequently, because the IRR relies on an incomplete honour system prone to both data stagnation and intentional nondisclosure [17, 19], these generated filters frequently fail to prevent modern route leaks, as demonstrated in major global routing incidents [4, 12, 13].

To address these inherent limitations of database-driven filtering and move towards real-time protection, more advanced and automatic cryptographic frameworks have been developed.

The first of these is **BGPsec** [20]. BGPsec was intended to provide end-to-end path validation, by having each AS cryptographically sign every advertised route as it travels down the path. However, hurdles like its high computational overhead and the need for "full-path" adoption have prevented the industry from fully embracing it, instead preferring more incremental solutions.

One of these incremental solutions as mentioned in the introduction is Resource Public Key Infrastructure - Route Origin Validation (RPKI-ROV) [5]. RPKI-ROV mitigates route hijacks by using a distributed system (the RPKI) to associate prefixes with AS numbers through Route Origin Authorizations (ROAs). This way, when an AS receives a new route advertisement, it is relatively simple to determine if the AS at the end of the path is actually authorised to originate the prefix.

While RPKI-ROV has seen significant global adoption, it is fundamentally limited to verifying the *origin* of the route [5, 21]. It cannot detect if the intermediate hops in the AS_PATH are legitimate nor if a route has been leaked or hijacked. This critical gap in path security is what the **Autonomous System Provider Authorization (ASPA)** protocol is designed to bridge.

3.3 Autonomous System Provider Authorization (ASPA)

ASPA (as defined in [6]) introduces a new type of object into the RPKI that allows an AS to cryptographically sign a list of its authorised providers. This collection of attestations can then be used to validate that every hop in a received AS_PATH is permitted according to the valley-free routing model described in the RFC [6].

3.3.1 Path Verification Algorithm

ASPA path verification algorithm conceptualises the AS_PATH as a series of ASes consisting of an up-ramp followed by an (optional) down-ramp, as illustrated in Figure 1. The verification consists of the following steps:

1. **Obtain ASPA objects**
   For each AS that is seen in the AS_PATH, the daemon retrieves the corresponding ASPA object from the RPKI repository, usually this is done through the Relying Party. If no ASPA is found, due to it not existing, or being cryptographically invalid, the result for that hop is **No Attestation**.

2. **Identify the up-ramp**
   Starting from the origin AS ❶ (the rightmost AS that originated the route) and moving leftwards, the sequence of hops in which each current AS lists the next AS as an authorised provider is the *up-ramp* ❷.

3. **Detect the transition to the down-ramp**
   When a hop is encountered where the current AS's ASPA object does not contain the next AS ❸, the up-ramp ends and the algorithm switches to the *down-ramp* ❹ phase. Once the algorithm switches over to the down-ramp phase, no more $Provider \leftarrow Customer$ hops should be encountered, as to not violate the valley-free routing principle. An example of a violating hop can be seen in Figure 1 ❺.

4. **Validate each hop**
   The daemon checks the ASPA object of the customer AS. In up-ramp mode, the relation is $Provider \leftarrow Customer$. In down-ramp mode, the relation is $Customer \rightarrow Provider$. If the next AS (following the directions above) is listed as a provider, the hop yields **Provider+**. If the next AS is not listed as a provider, the hop yields **Not Provider+**.

5. **Combine hop results** If any hop returns **Not Provider+**, the AS_PATH is

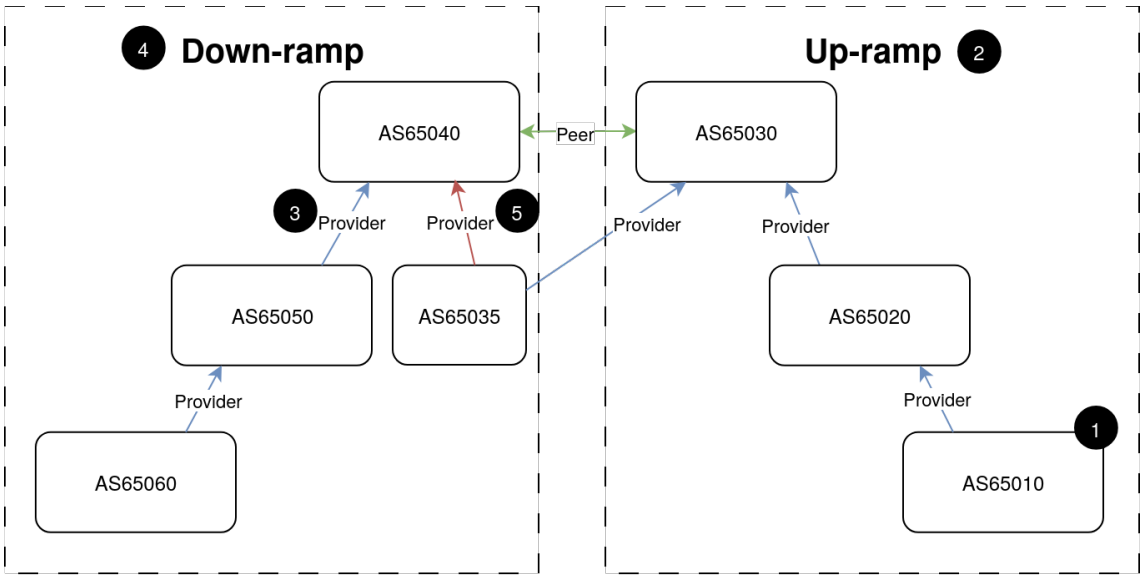

Figure 1: Example AS_PATH to be Evaluated by ASPA.

classified as **Invalid**. If all hops return **Provider+**, the path is **Valid**. Otherwise, the path is classified as **Unknown**.

3.3.2 Upstream vs. Downstream Validation Mode

The ASPA RFC defines two different validation modes: upstream mode and downstream mode. In upstream mode, the BGP daemon only expects to receive advertisements from customers (it considers itself an upstream only). Thus, it will reject advertisements in which the AS_PATH contains a down-ramp. In downstream mode, on the other hand, the daemons accept advertisements with AS_PATH containing both an up-ramp and a down-ramp.

4 Related Work

This section reviews existing literature evaluating ASPA alongside other path validation strategies.

4.1 Topological and Simulation-Based Effectiveness of ASPA

A significant portion of path validation literature focuses on evaluating the theoretical security benefits of ASPA through large-scale graph simulations. Furuness et al. used topological modelling to analyse the proposal against emerging post-ROV exploits [22].

Their findings demonstrated that ASPA delivers robust, cascading protection against accidental route leaks and malicious path manipulations under partial adoption states, even without immediate participation from Tier-1 networks [22].

Similarly, Umeda, Kimura, and Yanai evaluated deployment dynamics on real-world regional topologies using a custom route-exchange simulation. Their modelling quantified the security returns of partial deployments, reinforcing that strategic adoption at prominent intermediate network tiers can restrict malicious route propagation to under 10% of total victim networks [23].

4.2 Alternative Path Verification Frameworks

Other researchers have approached path insecurity through non-cryptographic mechanisms. For scenarios where cryptographic repository infrastructure remains sparse, such as the current early deployment phase where global ASPA adoption sits at roughly ±2% [21], Hao et al. developed *PathProb*, a probabilistic inference framework. This mechanism scores path legitimacy strictly based on historical routing data, offering an out-of-band means to flag anomalies [24].

Additionally, industry surveys compiled by Scott, Johnstone, and Szewczyk show the trade-offs between complex cryptographically-heavy frameworks and simpler options like the Only-To-Customer (OTC) attribute. Their survey outlines how these varying defensive strategies balance the detection of sophisti-

cated path forgery against the practical processing boundaries of the hardware on which these countermeasures would run [25].

4.3 Operational Constraints and Convergence Scaling

Introducing additional validation algorithms into active routing infrastructure introduces complex trade-offs with respect to hardware performance. Devikar, Patil, and Chandraprakash surveyed the elements that dictate BGP convergence delays and routing stability. Their analysis emphasised that edge routers are already pressured by vast Routing Information Base (RIB) and Forwarding Information Base (FIB) scaling limits [26]. Because real-world instability is frequently driven by a minor subset of hyperactive, "noisy" prefix updates, understanding the explicit state-management and computational overhead introduced by any new verification logic is fundamental to preserving stability and resilience [26].

4.4 Research Gap

While existing studies provide valuable insight into the theoretical security benefits of ASPA and its performance through simulations, there is a distinct gap in empirical research. Current literature relies heavily on abstract graph simulations, discrete-event models, or policy-based topological inferences. Consequently, there is a lack of real-world data examining how actual (open-source) BGP daemons handle these complex validation algorithms in practice.

5 Methodology

To address the aforementioned research by evaluating the performance and correctness of ASPA implementations across different BGP daemons, a containerised test harness was developed. This section outlines the experimental design, beginning with the core architectural requirements established to guide the development of the evaluation framework. Subsequent subsections detail the implementation of this architecture, the creation of synthetic and real-world-derived routing workloads, and the specific metrics measured across the experiments to address the core research questions.

5.1 Requirements

To ensure the test harness yields meaningful, accurate, and reproducible results, a set of core design requirements was formulated. These requirements define the capabilities and constraints the testing framework must satisfy in order to effectively address the underlying research objectives:

R1 Validity of Workloads: The workloads utilised in the testing process should reflect real-world usage scenarios. Workload validity ensures that the obtained validation results and performance measurements accurately reflect the behaviour of networking daemons in the real world.

R2 Relevance of Metrics and Experiments: The testing system must evaluate relevant metrics in order to provide fair and insightful comparison.

R3 Fairness: Fairness in testing is crucial to ensure an unbiased assessment of daemons. The testing process should provide a level playing field for all daemons, avoiding scenarios that may favour or disadvantage any particular daemon unfairly. Fairness promotes trustworthiness and reliability in testing outcomes.

R4 Clarity: The testing system should adopt a clear and structured approach to present results. Clear presentation of results aids in understanding and interpreting consistency and performance characteristics effectively, facilitating informed conclusions.

R5 Reconfigurability: The testing system should offer easy reconfigurability to enable testing new daemons.

R6 Ease of use: The testing system should prioritise ease of use, configuration, and setup procedures. User-friendly interfaces, documentation, and streamlined processes contribute to making the testing system accessible. This way, obtaining performance and validation results for new ASPA implementations remains straightforward.

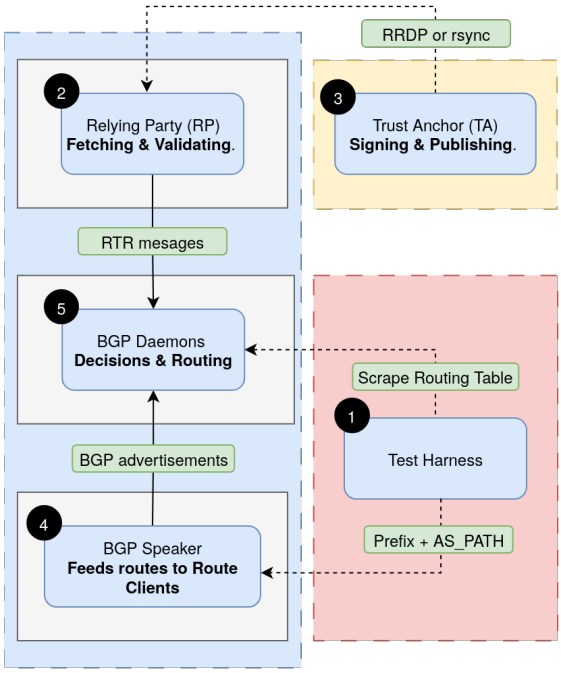

Figure 2: Test Harness Design

5.2 BGP Daemon Selection

To conduct the tests, it was essential to identify relevant BGP daemons that support ASPA. Since ASPA is an emerging standard, available options are currently limited. Among the major open-source BGP daemons, only BIRD and OpenBGPD were found to support ASPA validation at the time of writing.

- **BIRD:** BIRD Internet Routing Daemon is an open-source implementation for routing Internet Protocol packets on Unix-like operating systems. It was developed as a school project at the Faculty of Mathematics and Physics, Charles University, Prague.

- **OpenBGPD:** OpenBGPD, also known as OpenBSD Border Gateway Protocol Daemon is also open-source and built on Unix-like operating systems. The daemon is developed as part of the OpenBSD project.

Commercial options from vendors like Cisco and Juniper do not yet appear to support the standard. Furthermore, they are difficult to access without purchasing expensive hardware or signing restrictive licensing contracts, making them unfeasible to include in this evaluation.

5.3 ASPA Testbed Architecture

The infrastructure is orchestrated using Containerlab and Docker, ensuring that the test setup is reproducible, while remaining flexible. This allows later expansion of the testbed in order to add more daemons/routers for testing and comparison, directly satisfying **R5 (Reconfigurability)**, as new BGP Daemons can be introduced without altering the surrounding infrastructure. The containerised approach also supports **R6 (Ease of Use)**, since the entire testbed can be brought up or torn down from a single, version-controlled configuration rather than manual host setup. This architecture is outlined in Figure 2.

- **Test Harness ❶:** The test harness is responsible for inserting BGP advertisements into the test system and recording the resulting decision from the BGP daemons. Additionally, it collects timing information from the daemons to provide insight into their performance. Centralising both the insertion and the recording of results in a single component supports **R4 (Clarity)**, as all outcomes are captured and reported through one consistent interface rather than being scattered across daemon-specific logs.

- **Relying Party ❷**: The Relying Party (RP) fetches records from the configured Trust Anchors (TAs) ❸. This usually consists of the five Regional Internet Registries (RIRs), but can also be configured with other (self-hosted) TAs. The RP fetches the ROA and ASPA records from the configured TAs and perform cryptographic validation on them, thus reducing the processing overhead on the routers themselves. Once all the ROAs and ASPAs are processed it serves the valid records via the RPKI-to-Router (RTR) protocol to the BGP daemons. Serving identical, pre-validated ROA and ASPA data to every daemon from a single RP further reinforces **R3 (Fairness)**, ensuring that no daemon is disadvantaged by a slower or inconsistent RPKI validation path of its own.

- **BGP Speaker ❹:** A scriptable BGP injector used to present new advertisements to the test nodes. It announces BGP updates with varying AS_PATHs and prefixes, sourced from collected data or crafted datasets. It is fed BGP Routes

from the Test Harness ❶. The ability to script arbitrary, repeatable advertisement sequences addresses **R1 (Validity of Workloads)**, allowing both realistic and adversarial paths to be injected under controlled, repeatable conditions.

- **BGP Daemons ❺:** Multiple BGP daemons, which are compared against one another. The daemons access the RPKI infrastructure through the Relying Party and receive BGP advertisements from the BGP Speaker to perform validation.

5.4 Workloads

To thoroughly evaluate the BGP daemons, a diverse and realistic dataset was collected to feed into the testbed. This enables a robust comparison using two distinct approaches: historic real-world routing data and synthetically derived ASPA paths, together addressing **R1 (Validity of Workloads)** by grounding the evaluation in both observed and deliberately constructed traffic.

5.4.1 Historic Data

The historical datasets were extracted from public route collectors (e.g., RouteViews or RIPE RIS). These records were then used to evaluate how ASPA performs within the context of the current, sparse network of real-world ASPA adopters. Additionally, they provide insight into which existing routes would be categorised as invalid under ASPA deployment today. Grounding a portion of the workload in observed, real-world announcements is central to satisfying **R1**, as it ensures the daemons are also assessed against traffic patterns they would realistically encounter in a real-world environment.

5.4.2 Derived Data

To complement the historic data, synthetic datasets were derived from the ASPA records already registered within the RIRs. By constructing artificial paths of varying lengths, it is possible to simulate advertisements as they would be produced by a network with higher ASPA deployment. To further test the validation logic, intentionally invalid paths are introduced, adding non-ASPA nodes or incorrect provider relationships to verify that the daemons correctly reject invalid paths. These derived, edge-case paths extend coverage toward **R2 (Relevance of Metrics and Experiments)**, since they exercise validation branches that sparse real-world adoption would not currently surface.

5.5 Performance Metrics

5.5.1 Route Acceptance and Verification

To evaluate the consistency between the daemons, the decisions for each prefix are collected by the test harness, which highlights any prefixes that produce a different outcome. Additionally, to evaluate whether the daemons adhere to the logic modelled in the ASPA RFC, hand-crafted edge cases were used for testing, and any rejections were hand-checked for confirmation. This directly targets **R2 (Relevance of Metrics and Experiments)**, as correctness against the RFC's specified behaviour is arguably the most fundamental metric for comparing ASPA implementations. To compare upstream and downstream modes the daemons are ran in each mode in order to validate that they agree on route decisions in both modes.

5.5.2 Route Processing Delay

To evaluate the performance of the daemons, with and without ASPA enabled, timestamps are kept from the moment each daemon receives the route, to the moment it completes its validation. These two events are handled by using network timestamps of incoming packets, and the update timestamps of each daemon's RIB (Routing Information Base) respectively.

To ensure a fair comparison, the daemons are provisioned with identical resources, allowing us to accurately baseline the performance delta between them, satisfying **R3 (Fairness)**. Measuring both with and without ASPA enabled also satisfies **R2**, since it isolates the specific performance cost attributable to ASPA validation rather than general daemon overhead.

5.6 Test Execution Workflow

To guarantee reproducibility and streamline the experimental pipeline, the input ingestion, verification, and testing are managed through a custom orchestration script acting as the test harness execution workflow, in line with **R6**

(Ease of Use): a single script drives the entire experiment, so new ASPA implementations can be evaluated without bespoke setup for each daemon.

1. **Input Ingestion:** The harness pushes advertisements to the daemons through the programmable BGP Speaker and waits for the RIBs of each daemon to populate with the expected count.
2. **Validation Verification:** The harness queries the respective Command Line Interface (CLI) of each daemon to collect the decisions.
3. **Performance testing:** During execution, the harness programmatically captures decision latency: The total time elapsed from when a daemon receives a BGP update, until the BGP daemon updates its RIB.

This staged, automated workflow also reinforces **R4 (Clarity)**, since ingestion, validation, and performance results are produced in a consistent, well-defined sequence rather than being collected ad hoc.

6 Implementation

This section discusses the implementation of the ASPA test harness. The project group containing the repositories involved in the paper can be found on GitLab [27].

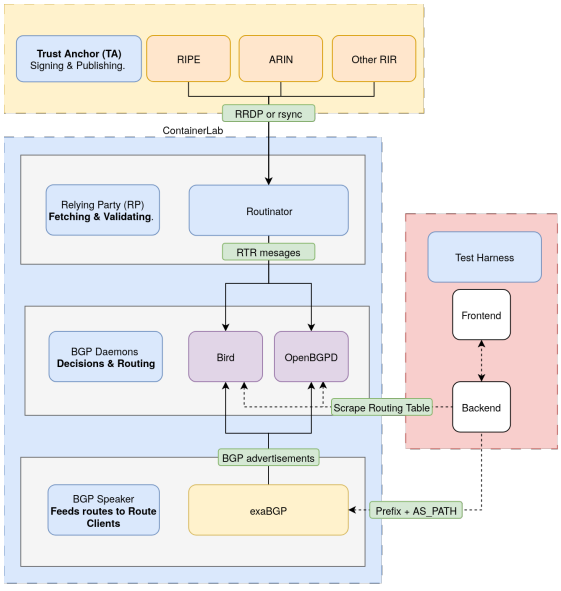

Figure 3: Implementation Architecture

6.1 Infrastructure

The infrastructure for the harness was built using Ansible to accomplish a repeatable setup. Additionally, Containerlab was used to create a virtual network to allow the containers to communicate with one another. Figure 3 shows how the final setup is structured and where all the pieces fit together. Since this is not a real world deployment, there was no need for separation of traffic. Thus, the Containerlab setup was built around one virtual switch for simplicity. The test harness accesses the containers through Containerlab's built-in management IP addresses.

6.1.1 Routinator

The Relying Party of choice was Routinator. Routinator is an open-source RP developed by NLnet Labs. It is one of the few RPs that provides support for ASPA, at the time of writing. Routinator was used with its default configuration, which uses the TAs of the five major RIRs to collect ROA and ASPA records. It also supports RTR, which is the current standard for sending the processed ROA and ASPA tables to the daemons.

6.1.2 ExaBGP

ExaBGP was used as the programmable BGP Speaker, as it provides a way of interacting with it through UNIX sockets. This allows the test harness to provide it with arbitrary prefix and AS_PATH combinations for ExaBGP to advertise to the daemons.

6.1.3 BIRD & OpenBGPD

Both containers were configured to store rejected routes, which would normally be discarded, into the filtered table for later retrieval. This can be accomplished by setting `import keep filtered` and `rde rib Loc-RIB include filtered` for BIRD and OpenBGPD respectively. Custom forks were utilised for both daemons, in which we added additional logging for more accurate timing. Logs were added before and after prefix handling, ROV validation and ASPA validation. This was necessary for OpenBGPD, since it only returns timestamps with a resolution in seconds, which is not precise enough. While BIRD does provide microsecond precision, it was found beneficial to also add identical logs here, for a more fair comparison.

6.2 Test Harness

The test harness was built as described in the Methodology. In order to improve the ease-of-use and convenience a web interface was built upon the backend. This backend performs all the operations related to actual tests: instrumenting ExaBGP to perform the advertisements and collecting the routes decisions from the daemons. The test harness was built so it does not have to run on the same system as the infrastructure itself. While this was mainly done for ease of development, it does also provide flexibility when the infrastructure is deployed on a remote system.

The interface enables the user to see the live status of the containers and run commands on them through the interface for debugging purposes. It additionally allows for advertising single routes for testing and verification. The main purpose of the interface is to enable the execution of tests. Tests can be run in two ways, either by specifying a list of route collectors and a timeframe, or by providing a JSON list of prefix and AS_PATH pairs. When a test is executed, the interface also enables the user to view and filter the results, for example on final decision or ASPA validation state.

6.2.1 Route Injection

Route injection makes use of the `/run/exabgp.in` socket. By piping in a list of commands into this socket, the harness can utilise ExaBGP to announce arbitrary routes to the daemons. ExaBGP does not modify the AS_PATH in order to faithfully replay the collected or crafted data. The commands used to advertise the routes are shown in A.3.

6.2.2 Collecting Route Decisions

Decisions on advertised prefixes are collected using each daemon's respective CLI. This was done by the backend of the test harness over SSH.

Because OpenBGPD does not allow explicit configuration of upstream/downstream mode, instead choosing this based on the neighbour definition, the validation logic was extracted from the source code. In order to get more complete results, the same was done for BIRD as well. These implementations were translated into equivalent Python functions, which allows the interface to display the result of both daemons in both modes for every tested advertisement.

The commands used for route decision extraction can be found in the appendix in sections A.1, A.2.

6.2.3 Collecting Timing

Timing data is collected by utilising forks of both daemons, with additional timing logs. Timestamps are logged when a prefix is received, when ROA validation starts and when ASPA validation starts. At the end of each of these steps, the current time is collected again to determine the duration of each step.

Both daemons utilise an internal monotonic clock for computing timestamps and the differences between them. Monotonic clocks advance at a uniform rate, providing greater accuracy when calculating elapsed time. Operating system clocks, on the other hand, prioritise time correctness, which can introduce jumps due to NTP corrections.

6.3 Workloads

To generate a dataset of valid and invalid ASPA paths, a script was developed that constructs ASPA paths based on existing ASPA registrations. The script produces a set of valid paths according to our understanding of ASPA, while intentionally relaxing strict rule adherence to introduce additional randomness into the generated paths. To produce invalid paths, the script first constructs a valid path and then deliberately breaks it by inserting a bad AS, causing the path to violate ASPA rules. To keep the process manageable, the dataset exclusively uses private-range IPv6 prefixes. This avoids the need to provide the daemons with valid ROV data for every generated path. Instead, since there exist no ROA records for private-range addresses, the daemons will mark them as `Unknown/NotFound` and proceed with ASPA validation. The final datasets collected and their purpose are outlined in Table 1.

7 Results

7.1 Consistency (RQ1)

When analysing the results of all 4 datasets (762, 234 total advertisements), it was found that both daemons agreed on what was valid and what was not. The results for this and the total advertisements per dataset can be seen in Table 2.

Dataset	Purpose
Captured Set	Small captured set for performance testing
Large Captured Set	Large captured set for validation
Valid Set	Artificial set filled with Valid ASPA paths
Invalid Set	Artificial set filled with Invalid ASPA paths

Table 1: Datasets and their Purpose

Dataset	# Of advertisements	Range of AS_PATH lengths
Captured Set	37,966	1 – 68
Large Captured Set	707,216	1 – 69
Valid Set	8,526	2 – 10
Invalid Set	8,526	2 – 10

Table 2: Total Number of Advertisements per Dataset

Additionally, all advertisements produced the same validation statuses for both ROV and ASPA. This shows that the ASPA implementations of both daemons are consistent with one another. As will be discussed in Section 7.2, ASPA daemons can be run in two different modes. The results show that the daemons reach the same conclusions, no matter which mode both are ran in.

7.2 Behaviour (RQ2)

On first inspection of these results, a suspiciously high number of rejections was observed, as can be seen in Figure 4. Given that ASPA is designed to support partial adoption, a rejection rate of over 25% was concerning. Upon reviewing the ASPA RFC and the source code of the daemons, the cause became clear: ASPA validation operates in two distinct modes, upstream and downstream. In upstream mode, the validating daemon expects routes only from customers, and therefore only accepts AS_PATHs consisting solely of an up-ramp. In downstream (or up-down) mode, it accepts AS_PATHs containing both an up-ramp and (optionally) a down-ramp. As noted, upstream mode produced a rejection rate of roughly 25% whereas downstream mode reduced this to 0.1%. The remaining rejections in downstream mode were all attributable to ASes listing ASPA records that do not include the next AS in the AS_PATH. As this is precisely the behaviour ASPA is intended to detect and reject, this result is a positive indication of correct validation.

It is important to note that while both daemons support upstream and downstream validation modes, BIRD forces the user to explicitly select which one should be used when configuring an ASPA filter rule, whereas OpenBGPD will select a validation mode based on the defined relationship with the peer in question.

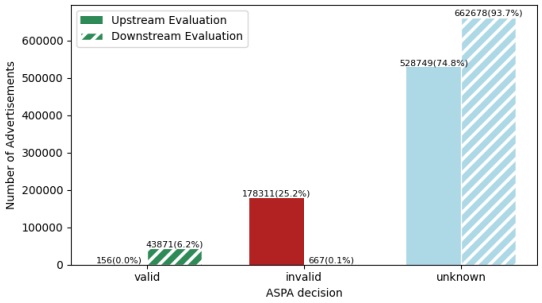

Figure 4: ASPA Validation Outcomes per Mode for the Large Collected Dataset. When in upstream mode, a large number of invalid paths is seen (25.2%) with very few valid paths. Downstream mode has more valid paths with very few invalid paths

7.3 Performance (RQ3)

To compare performance, the total processing time of each daemon with ASPA enabled and disabled was plotted. This can be seen in Figure 5.

7.3.1 Total Timing

In the case of BIRD, there is a minor increase across all test datasets when enabling ASPA.

Generally, each prefix appears to take only around 5 microseconds longer, with BIRD performing best on the collected dataset and showing the same performance for the invalid and valid sets. This is likely due to sparse ASPA coverage leading to less total processing time. Overall, the results are very consistent, with minimal spread.

OpenBGPD appears quite consistent, no matter whether ASPA validation is enabled or disabled. After further investigation, it was found that even when no ASPA validation rule was present in OpenBGPD's configuration and no ASPA records were provided, it would still execute the ASPA validation logic, even though these validation results would not be used for route decisions. The validation logic mainly consists of checks related to ASPA validity, such as: is the peer an eBGP session, is the peer's address IPv4 or IPv6 and is the peer a provider or customer? This is important to note as everyone running newer versions of OpenBGPD will pay the ASPA performance penalty, even if they do not wish to use ASPA. All three datasets share similar results, with only minor deviation between them.

7.3.2 ASPA Timing

When looking purely at the timing for ASPA validation, a different picture emerges as seen in Figure 6. Both BIRD and OpenBGPD appear extremely consistent, with similar results across datasets, except for the captured set. The captured set is interesting: BIRD's average timing is reduced, as expected, due to the sparsity of ASPA records compared to the other datasets. What is notable, however, is that OpenBGPD shows even better performance. For most of the $\pm 38,000$ prefixes, evaluation took exactly 4 microseconds. This seems unusual at first glance, but a closer look at the source code likely explains why. OpenBGPD utilises caching, unlike BIRD (at the time of writing), which stores ASPA validation results keyed with a combination of the AS_PATH and other BGP attributes such as origin, MED and local-pref. This caching is particularly beneficial for the captured set, as the nature of the data means certain AS_PATHs appear very frequently, resulting in less time spent accessing the lookup table.

When comparing these timings to ROV, ASPA performs better, taking on average 1 microsecond less per prefix. It is worth noting, however, that the internal ROA table is considerably larger than the ASPA table.

One limitation to note regarding the comparison of ASPA and ROA timings is that the maximum resolution provided by the daemons is 1 microsecond. This means that while the data appears well-distributed, it is distorted by a measurement accuracy of only ± 0.5 microseconds.

7.3.3 ASPA Performance vs. Path Length

To further analyse the performance of ASPA, processing time was also plotted against AS_PATH length to examine how it scales when handling longer paths. To do this, the captured dataset was used once more, as shown in Figure 7.

Some clear trends are visible: between AS_PATH lengths of 1-15, performance is fairly consistent for both BIRD and OpenBGPD, with a steady increase as expected. Unlike the results from Figure 6, here OpenBGPD performs worse than BIRD, taking approximately one microsecond longer to process, and also scaling less favourably, as indicated by its considerably steeper trend line.

It should be noted that the average path length in this dataset is 4, meaning longer paths are under-represented and results at the higher end are therefore less reliable. This is especially true for data to the right of the vertical line in Figure 7, where only 0.26% of samples fall. This largely explains the erratic behaviour observed in that region.

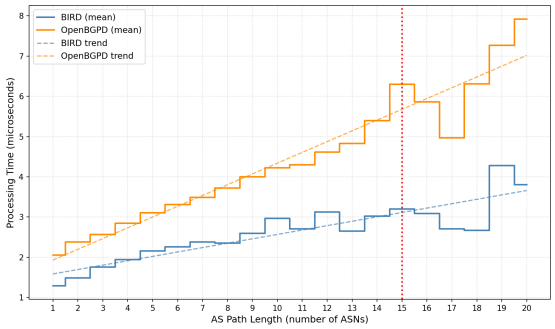

Figure 7: ASPA Validation Duration vs. AS_PATH Length per Daemon. The Figure shows a clear processing time increase when path length increases, with BIRD scaling better than OpenBGPD. When path length reaches 15, results become more erratic due to a lack of data

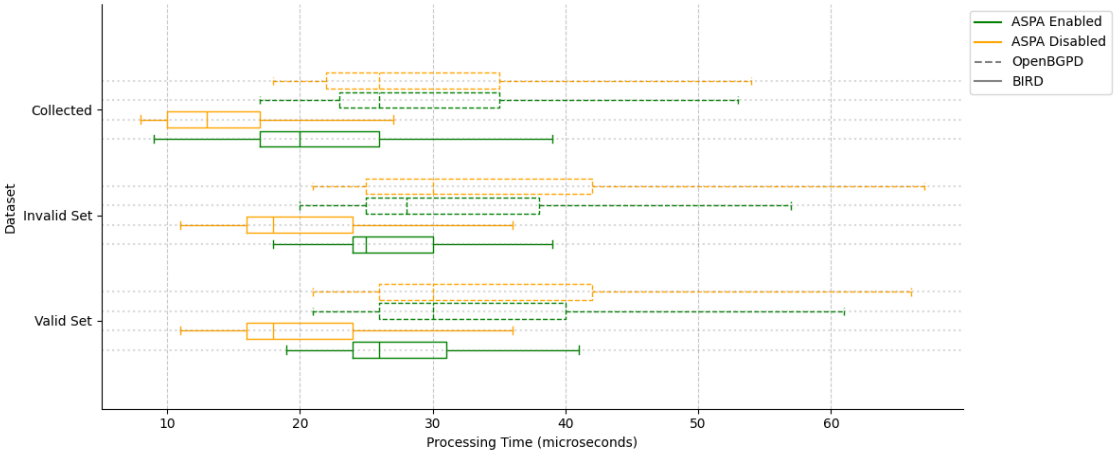

Figure 5: Total Prefix Processing time per Daemon with ASPA Enabled vs. Disabled. BIRD generally performs better than OpenBGPD, with ASPA only having a small impact

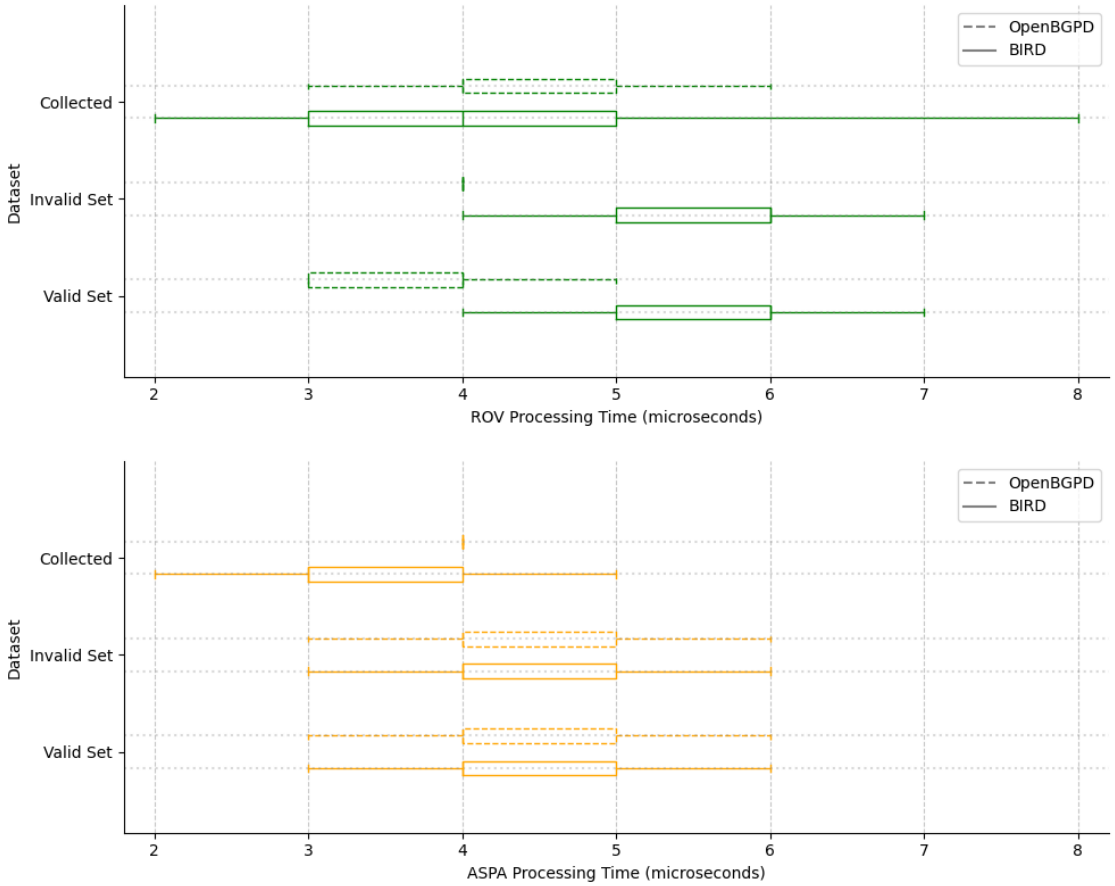

Figure 6: ROV (top) and ASPA (bottom) Processing Time per Daemon for each Dataset. ASPA shows faster and more consistent processing times than ROV. The collected dataset shows the highest performance due to the sparsity of ASPA-involved advertisements

8 Discussion

While the experimental results offer crucial insights into the performance and consistency of early ASPA implementations, a comprehensive evaluation requires analysing the limitations of the testbed. This section discusses the practical limitations encountered during the implementation and data collection phases, particularly regarding global protocol adoption. It concludes by outlining future research to expand upon these findings as the ASPA ecosystem matures.

8.1 Limitations

This research does have some limitations. The first and most significant of which is the limited number of daemons that support ASPA, which makes it difficult to comprehensively test the consistency of ASPA implementations.

An additional limitation related to this is the fact that ASPA is still in early adoption, with an adoption rate of roughly 2% [21]. This greatly limits the amount of collected paths that are of interest, as only a subset of the AS_PATHs seen in the collected advertisements actually involve ASPA. The limitation is especially noticeable in the larger datasets where the percentage of AS_PATHs involving any ASPA records is low. Running this test at a later stage, as more ASes and Regional Internet Registries adopt and incentivise ASPA, and a larger adoption rate is reached, will provide much greater insight into ASPA.

Additionally, our collected test set represents only a limited snapshot in time, due to the sheer volume of route advertisements. A longer or more spread-out time-frame might provide more interesting edge cases to analyse. However, this was not feasible in this work due to the processing time involved with datasets of this size.

Furthermore, our crafted datasets did not include any explicitly created edge cases, meaning more complex scenarios were not covered. While edge cases may have been present in the collected data, without deep inspection and a ground truth, it is difficult to ascertain whether they were present at all, and if so, whether they were validated correctly. The crafted datasets were also hard-limited by the number of currently published ASPA records. In order to increase the size of these datasets, one would need to set up their own RPKI infrastructure to create manufactured ASPA records for the daemons.

Lastly, the validation logic in the covered daemons has not yet been validated in production. As a result, there may be logic errors or unoptimised operations that prevent proper insight into behaviour and performance. This is explicitly acknowledged in the BIRD source code, where it is stated that the validation will be further optimised in the future once ASPA adoption increases [28].

8.2 Future Work

To build upon the findings of this project, several avenues for future research are proposed:

- **Test Additional Daemons:** Evaluate upcoming ASPA implementations in widely deployed industry daemons, such as FRRouting (FRR), Juniper, and Cisco, utilising this framework as a means of comparison.

- **Evaluate Resource Efficiency:** Benchmark the internal memory footprint and CPU utilisation of the routing engines under tight resource limits. While the daemons in this project were not constrained to the point of running out of resources, testing under strict resource limits would reveal which implementation is more efficient. This is critical since real-world routers often operate with limited hardware resources.

- **Utilise Larger Datasets:** Capture a longer time-frame or repeat the experiment at a later date when ASPA adoption has increased to improve the representativeness of the results.

- **Model Complex Topologies:** Develop sophisticated crafted datasets modelling complex, real-world relationships between ASes, including partial ASPA deployment and mixed provider-customer topologies, to rigorously test edge cases.

- **Replay Historical Incidents:** Replay actual historical route leaks and hijacks, such as the 2018 Amazon Route 53 incident, to determine whether ASPA validation would have detected or mitigated them.

9 Conclusion

Out of the roughly 762K tested prefixes, no differing outcomes were observed. Thus, it can be concluded that, barring any egregious edge cases, the ASPA validation implementation in both daemons is consistent. This is encouraging for ASPA, as it demonstrates that the draft, while complicated, does result in consistent implementations.

While the behaviour of the ASPA implementations appears to be consistent with the specifications, it was found that running the daemons in upstream vs. downstream mode has a significant impact on the final route decisions. This is especially notable given that the default mode (upstream) led to a rejection rate of 25%, which would cause serious disruption to the internet. As such, any AS adopting ASPA should be mindful of using the appropriate validation mode, particularly for ASes that have complex relationships with their neighbours.

Lastly, our performance testing shows that ASPA validation introduces a similar amount of processing time as ROV. This is a positive sign for ASPA adoption, as additional processing time could significantly hinder its adoption. Additionally, the performance impact appears to be fairly uniform, with per-implementation timings for ASPA being similar, aside from OpenBGPD's cache optimisation. That said, BIRD does perform better overall, which can be attributed to its greater emphasis on performance compared to OpenBGPD.

References

[1] K. Sriram et al. *Problem Definition and Classification of BGP Route Leaks*. RFC 7908. Section 2: Classification of Route Leaks, Pages 4-8. IETF, June 2016. URL: https://www.rfc-editor.org/rfc/rfc7908.html.

[2] Aftab Siddiqui. *BGP Hijack of Amazon DNS for Crypto-Theft*. Accessed: 2026-04-08. Apr. 2018. URL: https://www.internetsociety.org/blog/2018/04/amazons-route-53-bgp-hijack/.

[3] Angelique Medina. *Rostelecom Route Hijack Highlights BGP Security Vulnerabilities*. Accessed: 2026-04-08. Apr. 2020. URL: https://www.thousandeyes.com/blog/rostelecom-route-hijack-highlights-bgp-security.

[4] Tom Strickx. *How Verizon and a BGP Optimizer Knocked Large Parts of the Internet Offline Today*. Accessed: 2026-04-08. June 2019. URL: https://blog.cloudflare.com/how-verizon-and-a-bgp-optimizer-knocked-large-parts-of-the-internet-offline-today/.

[5] Pradosh Mohapatra et al. *BGP Prefix Origin Validation*. RFC 6811. Jan. 2013. DOI: 10.17487/RFC6811. URL: https://www.rfc-editor.org/info/rfc6811.

[6] Alexander Azimov et al. *BGP AS_PATH Verification Based on Autonomous System Provider Authorization (ASPA) Objects*. Internet-Draft draft-ietf-sidrops-aspa-verification-24. Work in Progress. Internet Engineering Task Force, Oct. 2025. 21 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-sidrops-aspa-verification/24/.

[7] Yakov Rekhter, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. Jan. 2006. DOI: 10.17487/RFC4271. URL: https://www.rfc-editor.org/info/rfc4271.

[8] Sandra L. Murphy. *BGP Security Vulnerabilities Analysis*. RFC 4272. Jan. 2006. DOI: 10.17487/RFC4272. URL: https://www.rfc-editor.org/info/rfc4272.

[9] Nils Höger, Nils Rodday, and Gabi Dreo Rodosek. "Mitigating BGP Route Leaks With Attributes and Communities: A Stopgap Solution for Path Plausibility". In: *International Journal of Network Management* 35.2 (2025). e70002 nem.70002, e70002. DOI: https://doi.org/10.1002/nem.70002. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.70002. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.70002.

[10] Cameron Morris, Amir Herzberg, and Samuel Secondo. "BGP-iSec: Improved Security of Internet Routing Against Post-ROV Attacks". In: Nov. 2023. DOI: 10.14722/ndss.2024.241035.

[11] Lixin Gao and Jennifer Rexford. "Stable Internet routing without global coordination". In: *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '00. Santa Clara, California, USA: Association for Computing Machinery, 2000,

pp. 307–317. ISBN: 1581131941. DOI: 10.1145/339331.339426. URL: https://doi.org/10.1145/339331.339426.

[12] Angelique Medina. *Ep. 35: Major BGP Route Leak Disrupts Internet Traffic Globally*. Accessed: 2026-04-08. Apr. 2021. URL: https://www.thousandeyes.com/blog/internet-report-episode-35.

[13] Alessandro Improta and Luca Sani. *BGP Leak by Vodafone Italy Disrupts Global Services*. Accessed: 2026-04-08. Apr. 2021. URL: https://www.catchpoint.com/blog/vodafone-idea-bgp-leak.

[14] Sandy Murphy et al. *Routing Policy System Security*. RFC 2725. Dec. 1999. DOI: 10.17487/RFC2725. URL: https://www.rfc-editor.org/info/rfc2725.

[15] APNIC. *The Internet Routing Registry (IRR)*. Accessed: 2026-07-06. 2023. URL: https://www.apnic.net/manage-ip/apnic-services/routing-registry/.

[16] RIPE NCC. *Internet Routing Registry Webinar*. Accessed: 2026-07-06. 2025. URL: https://www.ripe.net/documents/4082/IRR-Webinar-Slides_OrknEDY.pdf.

[17] Tobias Striffler. *The IRR Landscape: Data Quality - the Good, the Bad, and the Outdated*. Accessed: 2026-07-06. RIPE Labs. 2025. URL: https://labs.ripe.net/author/tobias-striffler/the-irr-landscape-data-quality-the-good-the-bad-and-the-outdated/.

[18] Stavros Konstantaras. *The IRR Landscape: Where Do ASes Keep Their Routes?* Accessed: 2026-07-06. RIPE Labs. 2024. URL: https://labs.ripe.net/author/stavros-konstantaras/the-irr-landscape-where-do-ases-keep-their-routes/.

[19] Savvas Kastanakis et al. "Replication: 20 Years of Inferring Interdomain Routing Policies". In: *Proceedings of the 2023 ACM on Internet Measurement Conference*. IMC '23. Montreal QC, Canada: Association for Computing Machinery, 2023, pp. 16–29. ISBN: 9798400703829. DOI: 10.1145/3618257.3624799. URL: https://doi.org/10.1145/3618257.3624799.

[20] Matt Lepinski and Kotikalapudi Sriram. *BGPsec Protocol Specification*. RFC 8205. Sept. 2017. DOI: 10.17487/RFC8205. URL: https://www.rfc-editor.org/info/rfc8205.

[21] Hurricane Electric. *RPKI & ASPA Adoption Report*. https://bgp.he.net/report/rpki_and_aspa. Accessed: 2026-07-06. 2026.

[22] Justin Furuness et al. "Securing BGP ASAP: ASPA and other Post-ROV Defenses". In: Jan. 2025. DOI: 10.14722/ndss.2025.240675.

[23] Naoki Umeda, Taiji Kimura, and Naoto Yanai. "The Juice Is Worth the Squeeze: Analysis of Autonomous System Provider Authorization in Partial Deployment". In: *IEEE Open Journal of the Communications Society* PP (Jan. 2023), pp. 1–1. DOI: 10.1109/OJCOMS.2022.3233833.

[24] Yingqian Hao et al. "PathProb: Probabilistic Inference and Path Scoring for Enhanced and Flexible BGP Route Leak Detection". In: Jan. 2026. DOI: 10.14722/ndss.2026.241691.

[25] Ben A. Scott, Michael N. Johnstone, and Patryk Szewczyk. "A Survey of Advanced Border Gateway Protocol Attack Detection Techniques". In: *Sensors* 24.19 (2024). ISSN: 1424-8220. DOI: 10.3390/s24196414. URL: https://www.mdpi.com/1424-8220/24/19/6414.

[26] Rohit Devikar, D. Patil, and V. Chandraprakash. "Study of BGP Convergence Time". In: *International Journal of Electrical and Computer Engineering (IJECE)* 6 (Feb. 2016), p. 413. DOI: 10.11591/ijece.v6i1.pp413-420.

[27] SNE RP2 ASPA Project Team. *ASPA Evaluation Test Harness Framework*. https://uva-hva.gitlab.host/groups/sne_rp2_aspa. Accessed: 2026-07-06. 2026.

[28] Bird. *Bird rt-table.c*. Accessed: 2026-07-03. URL: https://github.com/CZ-NIC/bird/blob/master/nest/rt-table.c#L350.

A Appendix

A.1 BIRD Commands

Accepted routes:

```
birdc show route protocol peer1 all
```

Rejected routes:

```
birdc show route filtered protocol
    ↪ peer1 all
```

Routes with invalid ROA:

```
# IPv4
birdc show route filtered protocol
    ↪ peer1 all where net.type =
    ↪ NET_IP4 && roa_check(t_rpki,
    ↪ net, bgp_path.last) =
    ↪ ROA_INVALID

# IPv6
birdc show route filtered protocol
    ↪ peer1 all where net.type =
    ↪ NET_IP6 && roa_check(t_rpki6,
    ↪  net, bgp_path.last) =
    ↪ ROA_INVALID
```

Routes with invalid ASPA:

```
birdc show route filtered protocol
    ↪ peer1 all where aspa_check(
    ↪ t_aspa, bgp_path, true) =
    ↪ ASPA_INVALID
```

Routes with valid ROA:

```
# IPv4
birdc show route filtered protocol
    ↪ peer1 all where net.type =
    ↪ NET_IP4 && roa_check(t_rpki,
    ↪ net, bgp_path.last) =
    ↪ ROA_VALID

# IPv6
birdc show route filtered protocol
    ↪ peer1 all where net.type =
    ↪ NET_IP6 && roa_check(t_rpki6,
    ↪  net, bgp_path.last) =
    ↪ ROA_VALID
```

Routes with valid ASPA:

```
birdc show route filtered protocol
    ↪ peer1 all where aspa_check(
    ↪ t_aspa, bgp_path, true) =
    ↪ ASPA_VALID
```

All routes not matching these queries are considered unknown.

A.2 OpenBGPD Commands

Accepted routes:

```
bgpctl -j show rib in inet
bgpctl -j show rib in inet6
```

Rejected routes:

```
bgpctl -j show rib filtered inet
bgpctl -j show rib filtered inet6
```

OpenBGPD indicates the state of each route using the Origin Validation State (`ovs`) and ASPA Validation State (`avs`) fields. These fields have the following possible values:

- Origin Validation State
  - valid
  - not-found
  - invalid
- ASPA Validation State
  - valid
  - unknown
  - invalid

A.3 ExaBGP Commands

```
announce route {prefix}
    ↪ next-hop {next-hop_ip}
    ↪ as-path [ {ASN_1} {ASN_2}
    ↪ ...]
```