Inter-domain Routing with Extensible Criteria

Seyedali Tabaeiaghdaei*
Anapaya Systems AG
Zurich, Switzerland

Jelte van Bommel
ETH Zürich
Zurich, Switzerland

Marc Wyss
ETH Zürich
Zurich, Switzerland

João Luis Sobrinho
Instituto de Telecomunicações,
Instituto Superior Técnico
Lisbon, Portugal

Giovanni Barbiero†
UBS AG
Zurich, Switzerland

Giacomo Giuliari
Mysten Labs
London, United Kingdom

Ahad N. Zehmakan
Australian National University
Canberra, Australia

Adrian Perrig
ETH Zürich
Zurich, Switzerland

Abstract

With the rapid evolution and diversification of Internet applications, their communication-quality criteria are continuously evolving. To globally optimize communication quality, the Internet's control plane thus needs to optimize inter-domain paths on diverse criteria, and should provide extensibility for adding new criteria or modifying existing ones. However, current inter-domain routing protocols and proposals satisfy these requirements at best to a limited degree.

We argue that an inter-domain routing architecture with extensible routing criteria can be realized in path-aware networks, due to their stateless forwarding. We thus propose IREC, an inter-domain routing architecture for the SCION path-aware Internet architecture that enables path optimization with extensible criteria. IREC achieves this through parallel execution and real-time addition of independent route computations, together enabling end domains to express their desired criteria to the control plane. Through large-scale simulations with realistic Internet topologies, we show IREC's viability via implementation and emulation, and its negligible global cost compared to static routing protocols.

CCS Concepts

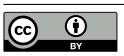
• **Networks** → **Routing protocols**; **Public Internet**; *Programmable networks*; *Programming interfaces*;

Keywords

SCION, routing protocols, routing on multiple optimality criteria, routing extensibility, inter-domain routing, path-aware networking, remote code execution

ACM Reference Format:
Seyedali Tabaeiaghdaei, Jelte van Bommel, Marc Wyss, João Luis Sobrinho, Giovanni Barbiero, Giacomo Giuliari, Ahad N. Zehmakan, and Adrian Perrig. 2025. Inter-domain Routing with Extensible Criteria. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3718958.3750528

*Most contributions while at ETH Zürich.
†All contributions while at ETH Zürich.



SIGCOMM '25, Coimbra, Portugal

ACM ISBN 979-8-4007-1524-2/2025/09
https://doi.org/10.1145/3718958.3750528

1 Introduction

The wide variety of today's networked applications requires a diverse set of communication quality criteria that need to be satisfied by the network for an optimal quality of experience. For example, video conferencing, Voice over IP (VoIP), and online gaming applications, each have specific criteria for communication quality. Furthermore, applications are rapidly evolving, and so are their communication quality criteria. For example, new criteria are expected to emerge with holographic communication [27], or the tactile Internet [24].

To meet these diverse criteria, the network must forward each application's traffic along paths optimized for its specific requirements, which necessitates discovering such paths within the network. Within the domain of wide-area network (WAN) providers, these optimizations have long been achieved through software-defined networking (SDN) and maintaining a complete, real-time view of the network. On the contrary, the public Internet, as an inter-domain network, has struggled to address the evolving needs of applications [31, 56]. This has driven the expansion of the private networks of hypergiants and the emergence of Network-as-a-Service (NaaS) providers. However, this trend has raised concerns about inheriting third-party policies and practices, vulnerability to outages, and industry consolidation [16].

The public Internet falls behind private networks primarily due to the stateful inter-domain forwarding and the limitations of the Border Gateway Protocol (BGP). Stateful forwarding restricts forwarding table sizes to the limited and expensive memory available on routers. Therefore, despite the existence of a super-exponentially large number of paths in the public Internet, routers usually only store a single path per destination in their forwarding tables. While this constraint limits the flexibility of route selection, the rudimentary and ossified route selection logic of BGP is also a consequence of operator choices, where paths are primarily optimized for monetary cost and hop length. Proposed changes to BGP for computing multiple paths per destination [25, 34, 36, 39, 41, 52, 57, 58] will also eventually hit the barrier of limited forwarding state in routers. Overlay architectures such as TANGO [16] also cannot find optimal

paths for diverse and evolving criteria as they can only discover a small number of paths and rely on BGP route computation.

Path-aware networking (PAN), on the other hand, takes a stateless approach to inter-domain packet forwarding by encoding the full inter-domain path in packet headers. This method eliminates the limitations imposed by stateful forwarding and BGP, with the control plane able to compute many paths between two Autonomous Systems (ASes). These paths are provided to endpoints (hosts within an AS), which can choose the most suitable one based on their criteria and the associated path metadata. The SCION [22] Internet architecture is a PAN commercially deployed by over 20 Internet service providers [23, 26, 30, 32, 48, 50]. In the current SCION network, though still relatively small yet expanding, we already observe that the current routing algorithm with a rudimentary path filtering computes a median of 100 paths between a pair of ASes, computing up to 400 distinct paths between some AS pairs. Note that this large number of paths is the number of computed paths by the control plane and not all paths. In the larger and denser topology of the Internet, the total number of paths can be super-exponentially large [35]. Lee et al. [16] count a median of around 5000 Gao-Rexford compliant paths between 1000 AS pairs randomly chosen from the CAIDA topology [20]. The number of paths can be even larger if we consider (1) the highest-degree ASes instead of random ones, (2) multiple connections between neighboring ASes, which are discoverable in SCION, and (3) the possibility of violating Gao-Rexford conditions in the core part of the SCION architecture [43].

The large number of paths will likely prompt the desire for even more optimality criteria, further increasing the importance of multi-criteria path optimization. Allowing endpoints to choose not only their local providers but also their wide-area provider [55], opens up the opportunity for network providers to convince more endpoints to steer their traffic through them to increase their revenue. Therefore, it is expected that providers introduce novel path metrics (e.g., the carbon intensity of paths [44, 49]) and advertise information regarding those metrics about segments of paths traversing their networks. Accordingly, the endpoints might update their optimality criteria to incorporate the novel metrics, accelerating the emergence of new optimality criteria.

While a promising opportunity, the explosive number of paths on the dense topology of the Internet poses considerable control-plane scalability challenges. Note that these scalability issues arise at a number of paths far exceeding those encountered in stateful forwarding. While PANs are capable of discovering and using thousands of paths per destination, stateful forwarding cannot go beyond a few. However, even PANs cannot support a combinatorial explosion in the number of paths.

In such circumstances, computing optimal paths for a specific criterion appears unattainable. First, due to the super-exponential overhead, it is not scalable to compute all paths and provide them to endpoints for optimization. Second, unlike a WAN provider's network, where optimal paths can be centrally computed, inter-domain routing is distributed across independent, competing entities, lacking a centralized map of the network. Third, optimizing paths on a fixed and arbitrary set of criteria (e.g., shortest AS-path length), which is oblivious to the criteria of end domains, can hide the optimal paths for a variety of other criteria and result in suboptimal performance.

However, scalable computation of optimal paths for various criteria is achievable if the inter-domain routing protocol optimizes paths *during route computation*. This approach allows routing nodes at each AS to only disseminate the optimal paths to neighboring ASes, avoiding the overhead of computing all paths. Given the wide variety of criteria, the routing protocol needs to *route on multiple optimality criteria*. Sobrinho et al. [47] lay the algebraic foundations for routing on multiple optimality criteria by proposing to route on the largest isotonic reduction of intersections of criteria (cf. Section 2). While being a major breakthrough in routing algebra, practical implementation of their principles remains missing, which is mainly due to the limitations of BGP. Furthermore, by focusing on the algebraic foundations, Sobrinho et al. do not answer the following operational questions that will appear in real-world deployments:

- *How to make inter-domain routing nodes distributed in different domains aware of the routing criteria on which they need to optimize paths?*
- *How to extend the routing criteria, i.e., how can new criteria efficiently be added, or old ones removed across the network?*

In this paper, we answer these questions by designing and implementing a new routing architecture for the deployed SCION PAN. Essentially, our architecture instantiates the principles by Sobrinho et al. [47] in an extensible manner from the operational viewpoint, minimizing the effort for introducing new routing criteria.

1.1 Obstacles to Criteria Extensibility

Our goal is to overcome the operational obstacles to the extensibility of inter-domain routing criteria. We thus highlight the major challenges.

Standardization.

The main obstacle to routing extensibility is the Internet Engineering Task Force's (IETF) time-consuming adoption of changes proposed by network operators, which hinders routing based on desired criteria. Even adopted changes take years to standardize, with BGP extension RFCs taking a median of 3.5 years to publish, and some taking up to 10 years [53]. This lengthy pace means standardization will likely lag behind the evolving optimality criteria of operators and applications.

Development and Deployment Cycle. Implementing inter-domain routing features is often expensive and time-consuming, requiring significant design and testing efforts, especially for hardware changes. Not all vendors or network operators adopt these standards, making updates tedious and resulting in incomplete deployments and suboptimal routing paths. xBGP [53] offers a vendor-neutral BGP architecture that facilitates the deployment of extensions in multi-vendor networks through an extended Berkeley Packet Filter (eBPF)-based API [3]. It advances routing extensibility but only within one domain, i.e., operators of one domain can change their BGP policies without waiting for standardization or vendor support. Therefore, it lacks cross-domain coordination mechanisms for introducing new routing criteria and does not provision for domains to express their desired routing criteria.

Inconsistent Implementation and Deployment across the Internet. Vendors may implement the same standard differently. For example, they use different algorithms for path selection based on the BGP MED attribute [6]. Routers from the same vendor across the network can have inconsistent software versions, possibly causing suboptimal or nondeterministic routing. For instance, a router with new software optimizing for latency and carbon emission may conflict with one optimizing only for latency. xBGP amplifies this issue by allowing each AS to create unique BGP extensions, increasing implementation variants.

Control-plane Interruption. Updating router software or hardware requires restarting routing sessions with neighboring routers, leading to route re-exchanges that may flood the network with routing messages. Frequent updates for optimality criteria can overwhelm some routers with an excessive number of routing messages.

Data-plane Interruption. Updating routers can disrupt the data plane by: (1) necessitating traffic rerouting to backups during restarts, (2) causing forwarding loops and blackholes due to inconsistent network views during control-plane convergence, and (3) overwhelming routers with heavy control-plane processing, disrupting their forwarding capabilities. Such interruptions are not relevant to PAN architectures as data and control planes are decoupled.

1.2 Contributions

We design IREC, i.e., **I**nter-domain **R**outing with **E**xtensible **C**riteria, a new routing architecture for the SCION PAN that enables (1) multi-criteria inter-domain routing, and (2) real-time and automated addition and removal of criteria. We build IREC on two key intuitions. First, separately optimizing paths for different sets of criteria in parallel, allowing the design of a flexible system that adds or removes criteria without interrupting other optimization processes. Second, allowing end ASes (destination or source of traffic) to communicate their routing criteria using the routing messages they originate, while other ASes—incentivized by receiving more traffic and increasing revenue—optimize routes according to the specified criteria. To realize these intuitions, we design a system to be deployed in participating ASes. The system uses eBPF technology [3] to automatically deploy and execute an arbitrary number of route selection programs *in parallel*. Every route selection program is an eBPF program that selects paths for a separate set of criteria. Each AS can independently deploy their desired selection programs. The second intuition is realized by using routing messages as a vessel to deliver the bytecodes of programs desired by end ASes and *automatically* deploy and execute them in all other ASes for the routes pertaining to the end AS.

We show IREC's viability through implementation and emulation on a realistic topology, where it achieves reductions in median CPU and memory usage compared to the existing monolithic implementation of the SCION control plane. Furthermore, using simulations on a much larger Internet topology, we examine the impact of routing with separate but smaller sets of criteria and compare it with that of routing with one larger set of criteria. We observe similar message complexity and convergence time, suggesting that IREC can achieve extensibility with negligible cost.

In summary, we contribute to a performant and extensible Internet by:

- designing IREC, a control-plane architecture for the SCION PAN, enabling extensible and multi-criteria path optimization (cf. Sections 4 and 5);
- implementing IREC in the open-source codebase [12] of the SCION PAN architecture [22] using user-space eBPF (uBPF) [14] (Section 6), and emulating it using the Kathará emulator [17]; and
- evaluating IREC on a realistic Internet topology by developing and using a simulator based on ns-3 [51] (cf. Section 7).

This work does not raise any ethical issues.

2 Background on Routing Algebras

Routing algebra provides the formal and mathematical foundation of routing protocols, and designing a sound routing protocol requires meticulous attention to its algebraic underpinnings. We build IREC on the solid routing algebraic foundations of multi-criteria path optimization laid by Sobrinho et al. [47]. Therefore, a basic knowledge of these concepts and constructs, which we provide in this section, is necessary to read and understand IREC. In this section, we represent these concepts in a less mathematical and more SCION-tailored language.

SCION beaconing is a path discovery protocol capable of enumerating all possible simple paths between AS pairs. However, the number of such paths may grow super-exponentially with the size of the network—it is $\Theta(n!)$, if every one of n ASes directly connects to every other [35]—making path discovery protocols fundamentally unscalable. In contrast, a multi-path routing protocol has ASes collectively select specific paths between AS pairs based on predefined criteria, filtering out all others in the process. Routing algebras provide the abstractions needed to transform a path discovery protocol into a general-purpose multi-path routing protocol. A *routing algebra* $(A, \preceq, \oplus)$ consists of a set A of *attributes* equipped with a *comparison relation* $\preceq$ and an *extension operation* $\oplus$. Attributes represent arbitrary routing metrics and/or policies. Each link in a network is assigned an attribute, and the attribute of a path is computed by successively applying the extension operation to the attributes of its constituent links. The comparison relation defines a preference between pairs of attributes, and thus between pairs of paths. A *selection operation* is then derived from the comparison relation to identify the most preferred attributes within a set. Routing algebras explore algebraic relationships between $\preceq$ and $\oplus$, remaining agnostic to the semantics of attributes. In scenarios where routing computations are initiated by destinations, one key algebraic property is left-isotonicity:

$$\forall a, b, c \in A,\ b \preceq c \Rightarrow a \oplus b \preceq a \oplus c.$$

This property states that the relative preference between two attributes is inherited by their left extensions with a common, third attribute. Left-isotonicity has strong implications for the design of routing protocols. Consider attribute a, associated with link uv, and attributes b and c, each associated with one of two paths from AS v to a common destination. Suppose that $b \prec c$, and that AS v selects optimal attribute b. Left-isotonicity determines that AS u prefers $a \oplus b$ over $a \oplus c$, or that the two extensions are equal. Therefore, as long as AS v advertises its optimal attribute to AS u, the latter will also select its own optimal attribute $a \oplus b$. Moreover,

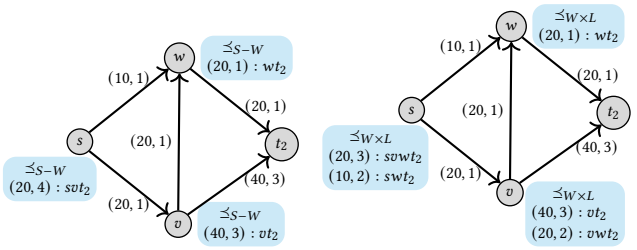
if AS v does not advertise non-optimal attribute c to AS u, then neither the extension of a with c nor the comparison between $a \oplus b$ and $a \oplus c$ are performed—saving one extension operation and one comparison. This atomic example illustrates how left-isotonicity paves the way to optimal and efficient routing protocols.

The earliest routing algebras were developed to reason about the routing protocols of the current Internet, where the comparison relation is a total order, multi-path routing is limited to Equal-Cost Multi-Path (ECMP), and left-isotonicity does not hold in general [46]. Sobrinho and Ferreira [47] addressed these limitations by lifting the comparison relation from a total order to a partial order, thereby allowing some pairs of attributes to be incomparable, with neither attribute of the pair preferred to the other. Particularly relevant is the possibility of transforming a total order that violates left-isotonicity into a partial order that satisfies it, by judiciously declaring certain pairs of attributes incomparable. Such a partial order is called a left-isotonic reduction.

To illustrate, consider the *shortest-widest* algebra consisting of pairs (w, l), where w is a width and l is a length, representing metrics such as capacity and delay, respectively. Widths extend with the minimum operator and lengths with addition. The comparison relation is the shortest-widest (total) order, denoted $\preceq_{S-W}$, where $(w, l) \preceq_{S-W} (w', l')$ if $w > w'$, or $w = w'$ and $l \leq l'$. This routing algebra is not left-isotone. In Figure 1a, every link is annotated with a pair width-length. AS v has two alternative paths to reach t_2: through link vt_2, with width-length $(40, 3)$; and through path vwt_2, with width-length $(20, 2) = (20, 1) \oplus (20, 1)$. AS v selects the former width-length because it has greater width, and advertises it to AS s. As a consequence, AS s learns only $(20, 4) = (20, 1) \oplus (40, 3)$ from v. It selects this width-length, which is not that of the shortest-widest path from s to t_2. The shortest-widest path from s to t_2 is path $swvt_2$ with width-length $(20, 3) = (20, 1) \oplus (20, 1) \oplus (20, 1)$.

Instead, consider the *product order* on width-lengths, denoted $\preceq_{W\times L}$, where $(w, l) \preceq_{W\times L} (w', l')$ if both $w \geq w'$ and $l \leq l'$. The product order is a *left-isotonic reduction* of the shortest-widest order that renders incomparable any two width-lengths where one has both greater width and greater length than the other. Figure 1b shows the same network as Figure 1a, but operated with a multi-path routing protocol that selects width-lengths according to the product order. AS v selects both $(40, 3)$ and $(20, 2)$, since these two width-length are incomparable. Consequently, AS s learns both $(20, 4) = (20, 1) \oplus (40, 3)$ and $(20, 3) = (20, 1) \oplus (20, 2)$ from v. It selects $(20, 3)$, which indeed is the width-length of the shortest-widest path from s to t_2, and filters out $(20, 4)$. Notably, the product order is also a reduction of the widest-shortest order, denoted $\preceq_{W-S}$, where $(w, l) \preceq_{W-S} (w', l')$ if $l < l'$, or $l = l'$ and $w \geq w'$. Thus, AS s also selects width-length $(10, 2)$, learned from AS w, which is that of the widest-shortest path from s to t_2. AS s has the ability to forward data-packets on either shortest-widest or widest-shortest paths, a user decision that may depend on the application.

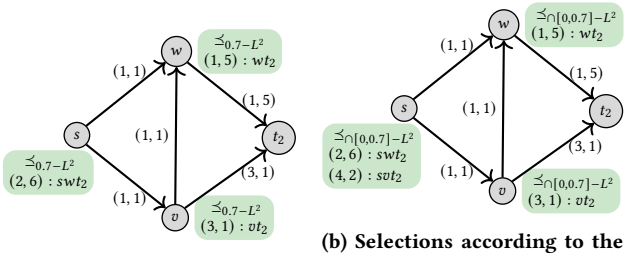
Alongside left-isotonic reductions, Sobrinho and Ferreira [47] introduce the concept of intersecting multiple orders into a single partial order. To illustrate, consider attributes given as pairs of lengths (l_1, l_2), where l_1 and l_2 represent metrics such as delay and carbon emission, respectively. The preferences of a network user among such pairs can be captured by the α-linear-combination of



(a) Selections according to the shortest-widest order.

(b) Selections according to the product order on width-lengths.

Figure 1: Examples of routing algebras on width and length. Links annotated with width-length.



(a) Selections according to 0.7-linear combination of the two lengths.

(b) Selections according to the intersection of the α-linear-combination of the two lengths for $\alpha \in [0, .0.7]$

Figure 2: Examples of routing algebras on pairs of lengths. Links annotated with pairs of lengths.

the lengths of a pair, defined as

$$L_\alpha(l_1, l_2) = \alpha \times l_1 + (1 - \alpha) \times l_2,$$

with $\alpha \in [0, 1]$. The induced comparison relation, $\prec_{\alpha-L^2}$, is such that $(l_1, l_2) \prec_{\alpha-L^2} (l'_1, l'_2)$ if $L_\alpha(l_1, l_2) < L_\alpha(l'_1, l'_2)$.[1] Different network users may have different choices for the value of α. Therefore, the network should be able to provide paths optimized over an entire range $[\alpha_m..\alpha_M]$ of values of α. This can be achieved by a comparison relation $\prec_{\cap[\alpha_m..\alpha_M]-L^2}$ that intersects all of the $\prec_{\alpha-L^2}$, for $\alpha \in [\alpha_m..\alpha_M]$. We have $(l_1, l_2) \prec_{\cap[\alpha_m..\alpha_M]-L^2} (l'_1, l'_2)$ if both $L_{\alpha_m}(l_1, l_2) < L_{\alpha_m}(l'_1, l'_2)$ and $L_{\alpha_M}(l_1, l_2) < L_{\alpha_M}(l'_1, l'_2)$. In Figure 2a, every link is annotated with a pair of lengths. According to the 0.7-linear-combination of the two lengths, AS v selects $(3, 1)$, learned from t_2, over $(2, 6)$, learned from w, since $L_{0.7}(3, 1) = 2.4 < 3.2 = L_{0.7}(2, 6)$. AS s learns $(2, 6)$ from AS w and $(4, 2)$ from AS v. It prefers $(2, 6)$ because $L_{0.7}(2, 6) = 3.2 < 3.4 = L_{0.7}(4, 2)$. Figure 2b shows the same network as Figure 2a, but operated with a multi-path routing protocol that selects all optimal pairs of lengths for every α-linear-combination with $\alpha \in [0..0.7]$. AS v still selects only $(3, 1)$, since $L_{0.7}(3, 1) < L_{0.7}(2, 6)$ and $L_0(3, 1) = 1 < 6 = L_0(2, 6)$. However, AS s now selects both $(2, 6)$ and $(4, 2)$, because although $L_{0.7}(2, 6) < L_{0.7}(4, 2)$, we now have $L_0(2, 6) = 6 > 2 = L_0(4, 2)$. A user of AS s requiring an optimal 0.7-linear-combination path to t_2 adopts the swt_2 path provided by the multi-path routing protocol,

[1] For simplicity of presentation, we let (l_1, l_2) and (l'_1, l'_2) be incomparable when $L_\alpha(l_1, l_2) = L_\alpha(l'_1, l'_2)$, rather than adding a tie-break between the two pairs.

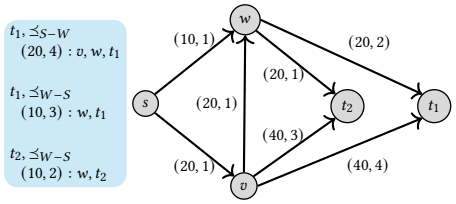


Figure 3: Example of multi-criteria path optimization.

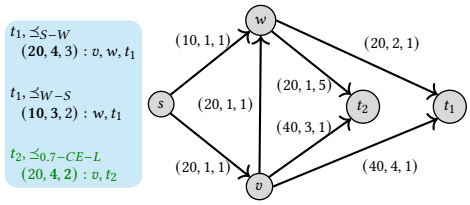


Figure 4: The need for extensibility of routing criteria.

whereas a user requiring an optimal fifty-fifty balance between the two lengths ($\alpha = 0.5$) adopts svt_2.

We use the algebraic concepts introduced in this section to instantiate realistic motivating examples for the design of IREC and to design the routing mechanisms of IREC. Therefore, IREC builds on the concepts and principles of routing algebras summarized above to implement an operational, flexible multi-path routing protocol for path-aware network architectures, supporting multiple optimality criteria.

3 Motivating Examples

Building upon the algebraic introduction provided in Section 2, in this section, we provide two motivating examples for designing IREC, which guide the reader through the design section, i.e., Section 4. In these examples, we introduce a real-world instantiation of the concept of path attributes introduced in Section 2: Attributes are tuples of elementary metrics, each of which is intrinsically associated with an extension operation. For example, a metric such as delay extends with addition, while a metric such as bandwidth extends with min. The extension operation associated with a given elementary metric is known, i.e., standardized a priori. The constitution of attributes as tuples of concrete metrics is also known, i.e., standardized, in advance to all nodes involved in routing computations, although it is also possible to introduce dynamism in that constitution. What changes dynamically among routing computations is the selection criteria on attributes, or, equivalently, the order among attributes.

3.1 Multi-Criteria Path Optimization

Figure 3 illustrates an inter-domain topology in which each node represents an AS and each edge represents an inter-domain link with attribute (w, l), where w is bandwidth in Gbit/s and l is latency in ms. Using this topology, we provide an example of the need for path optimization on different criteria for different applications and end domains. Suppose AS t_1 hosts two services: a video-call service to which customers should reach via paths primarily with high bandwidth and secondarily with low latency, and a VoIP service to which customers should reach via paths primarily with low latency and secondarily with high bandwidth. Concurrently, suppose that AS t_2 is hosting a VoIP application that also needs to be reached through primarily low-latency and secondarily high-bandwidth paths. The video call's optimality criterion is the shortest-widest order, $\preceq_{S-W}$. The VoIP's optimality criterion is the widest-shortest order, $\preceq_{W-S}$. The blue box at the left of node s in Figure 3 shows the desired final state of routing protocol at s for each destination t_1 and t_2. To reach this final state, the control plane needs to be aware of different routing criteria for t_1 and t_2 and optimize paths accordingly.

3.2 Criteria Extensibility

Suppose that link attributes in the topology can be augmented with a new quality metric, e.g., the carbon emission of paths in terms of carbon intensity of data transmission in g/(Gbit s) . Figure 4 illustrates the updated topology, where the link attribute (w, l, CE) represents bandwidth w, latency l, and carbon emission CE, which is also a length. With such a new metric, new optimality criteria are expected to emerge. For example, suppose the AS s wants to incorporate greenness into its path selection criterion to AS t_2 by defining a total order $\preceq_{\alpha-CE-L}$. An α value of 0.7 results in the change of the desired path to t_2 for s compared to the previous widest-shortest criteria. For this path to be advertised to s, v needs to be aware of this change of routing criteria (and possibly the value or the range of α at s) and update the routing logic for destination t_2 and source s accordingly. Importantly, the value of α can differ for different source and destination AS pairs, resulting in distinct criteria, all of which should be accommodated by the control plane.

4 IREC's Routing Mechanisms

In this section, we define our design goals and introduce IREC's routing mechanisms to achieve these goals: (1) Orthogonal route computations (RCs), (2) On-demand RCs and (3) Reverse On-demand RCs. Together, these mechanisms achieve all four goals of extensibility of routing criteria, as we describe next.

4.1 Design Goals

To realize the extensibility of inter-domain routing criteria in SCION, our design must overcome the challenges mentioned in Section 1.1. Therefore, we specify the following concrete goals, the achievement of which is essential for a design to overcome those challenges:

- **G1** Enabling the addition and removal of routing criteria without interrupting the routing on other criteria.
- **G2** Allowing each domain to develop and deploy routing criteria of their choice without waiting for standardization or vendors.
- **G3** Providing a mechanism for automated, real-time, and consistent deployment of new routing criteria in multiple domains without standardization, vendor support, or involvement of their network operators.
- **G4** Providing end domains, i.e., any domain that can be the source or destination of traffic, with a real-time mechanism to express their desired routing criteria with immediate effect, i.e., a new route computation according to the expressed criteria is instantly started.

4.2 Overview

IREC introduces two distinct routing mechanisms. The first mechanism, *orthogonal RCs*, enables multiple route computations (RCs) to run in parallel. Each of these RCs optimizes routes based on an independent set of criteria. Because these RCs are orthogonal to each other, i.e., they execute independently of each other, new criteria can be added or removed without affecting other RCs. Each RC uses a dedicated route selection program (SP) that defines its optimization criteria.

The second mechanism, *on-demand RCs*, builds on the independent SPs to provide an interactive interface through which end domains can express their desired routing criteria. In this mechanism, an AS embeds the bytecode of its desired SP directly within the routing messages it originates. This SP defines the route optimization criteria from the perspective of the originating AS. Motivated by the potential to receive more traffic and increase revenue (cf. Section 8), ASes other than the originating AS deploy and execute these SPs in real time – but only for the routes pertaining to the originating AS. This enables ASes to implicitly agree on the route optimization, without standardization, vendor support, or human involvement.

On-demand RCs enable route computations in *one direction*, using an SP specified by the originating AS, to optimize the paths that other ASes will use to reach it. The originating AS itself cannot directly use the paths discovered through this mechanism [2], unless additional steps are taken – such as explicitly requesting the paths from the receivers or having the receivers register them. However, these paths are not always desirable in the opposite direction, due to asymmetry the characteristics of a path may differ significantly from those of that same path in the reverse. Optimizing only in one direction with an SP can lead to suboptimal performance for bidirectional communication.

To address this limitation, we introduce our third mechanism: *Reverse On-Demand RCs*. This mechanism allows the originating AS to initiate a separate route computation that explicitly initiates path discovery in the reverse direction, i.e., from the originator to the remote ASes, according to a specified SP.

4.3 Orthogonal Route Computations

Orthogonal RCs are independent RCs computing paths using distinct selection operations without interfering with each other. This contrasts with the conventional routing, where a single RC computes routes using a unified selection operation.

The selection operation of an RC selects paths using *either* of the following methods:

- It sorts path attributes based on a total order that satisfies isotonicity, and selects the most preferred attributes or
- It sorts path attributes based on the largest isotonic reduction of the intersection of multiple total orders, and selects the sets of most preferred attributes [47].

We refer to the executable implementation of a selection operation as a *selection program (SP)*, which is executed by ASes participating in the corresponding RC.

[2] In SCION, any path can be reversed, such that it can be used for bidirectional communication.

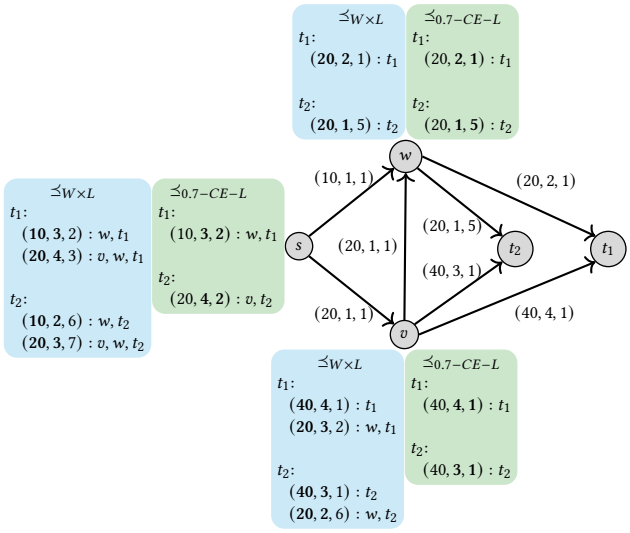


Figure 5: An example of orthogonal RCs.

Each SP is a plug-and-play and isolated program that takes into account a subset of performance metrics present in the path attributes. As an example, assume that the path attributes are in (w, l, CE), where w, l, and CE represent the bandwidth, latency, and carbon emission of a path, respectively. There can exist two parallel RCs: one with a selection operation that selects paths based on their bandwidth and latency (w, l), and another with a selection operation that only considers the latency and carbon emission (l, CE).

IREC achieves multi-criteria path optimization with a different approach than the one proposed by Sobrinho et al. [45]: As an alternative to routing on a *single* intersection of *all* criteria, IREC enables routing on *different subsets* of optimality criteria. This allows for the extensibility of routing criteria from an operational perspective. Nevertheless, IREC does not introduce new algebraic concepts but re-uses the ones introduced by Sobrinho et al. [45] in a new way.

4.3.1 Examples Revisited. Figure 5 re-uses the example in Figure 4 to show how orthogonal RCs work and optimize paths for multiple criteria in practice. In this example, nodes v, w, and s all deploy the same two SPs. The first SP computes the set of most preferred attributes according to the product order of bandwidth and latency ($\preceq_{W\times L}$) [29]. The second SP sorts attributes based on the linear combination of latency and carbon emission, i.e., $\alpha \times CE + (1 - \alpha) \times l$, where α is 0.7. As the example shows, the SPs that run in parallel may select the same attributes, which means redundant computation. This computational overhead is the inevitable price that needs to be paid for extensibility. However, bandwidth-use overhead can be avoided by an optimization where advertisements that would repeat the same path across different criteria are omitted. For example, if P is a selected path according to both criteria A and criteria B, then P and its attribute is advertised only once (cf. Section 5).

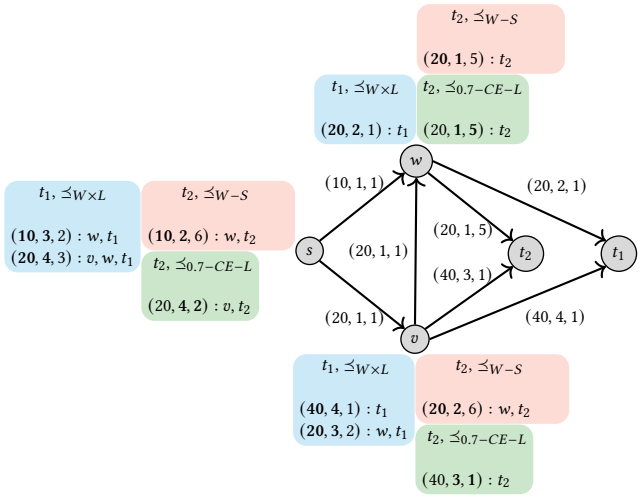


Figure 6: Examples of on-demand RCs.

4.4 On-demand RCs

On-demand RCs make use of the ability to add new independent RCs provided by the orthogonal routing mechanism. On-demand RCs enable each AS to independently express its desired optimization criteria for paths pertaining to itself, and ensure that other ASes optimize those paths according to these expressed criteria. Therefore, it eliminates the need for SP standardization, vendor support, or manual deployment efforts in other ASes. An on-demand RC optimizes paths *from any AS to the originating AS*. In a client-server model, this means optimizing the path from clients to the server.

To communicate the routing criteria, the AS develops and builds a corresponding SP into executable bytecodes (e.g., eBPF [3] or WebAssembly [15]) and serves it on a server within its AS. It then originates routing messages, so called path-construction beacons (PCBs),[3] which contain a new extension that specifies the URL of the SP. Motivated by increasing the probability of receiving more traffic and increasing revenue (cf. Section 8), every AS receiving such PCBs will likely fetch the SP from the originating AS, executes it on the set of PCBs specifying that same SP, and propagates the PCBs selected by the SP to its neighbors. Due to orthogonal RCs, these SPs are executed in parallel to other SPs, thus not interrupting or interfering with each other. To fetch the SP, the receiving AS can use any of the already received paths to the originating AS.

[3]In SCION, paths are computed by the beaconing protocol. The routing messages in beaconing are called path-construction beacons (PCBs.), PCBs are originated from a specific set of ASes (core ASes) and propagated through the network. ASes evaluate, select and disseminate the most suitable PCBs based on their routing criteria and policies. Each AS that disseminates a PCB, appends information about its AS hop, including the ingress and egress interfaces of the AS from which the PCB entered and exited the network. ASes can also add metadata about their AS hop, which can be used for path optimization. PCBs construct segments of SCION paths, and are registered at each AS's control service. To construct the full paths, endpoints request path segments from control services in the network, and concatenate them into complete paths, which can be encoded in packet headers. For more information on SCION routing please refer to Appendix E.

4.4.1 Examples Revisited. Figure 6 provides an example of an an on-demand RCs satisfying the criteria presented in Figure 3 and Figure 4 (cf. Section 3). In this example, there is no standardized generic RC and that all RCs are defined by destinations.

Some applications in destination t_1 prefer the widest-shortest paths, and some prefer the shortest-widest paths. However, the shortest-widest order is not isotone [45]. Therefore, destination t_1 starts an RC that computes the set of most preferred attributes to t_1 according to the product order of bandwidth and latency ($t_1, \preceq_{W\times L}$), which is the largest isotonic reduction of the shortest-widest order. This order also allows for routing on the widest-shortest order.

Destination t_2, on the other hand, initially prefers the widest shortest paths. Since the widest-shortest order is an isotone total order, destination t_2 starts an RC that sorts path attributes to t_2 using the widest shortest order ($t_2, \preceq_{W-S}$) and selects the best one. Later, when the carbon emission of the paths is introduced as a valid performance metric, the destination t_2 starts another RC that selects the attribute to t_2 with the lowest 0.7-linear combination of latency and carbon emission ($t_2, \preceq_{0.7-CE-L}$). If destination t_2 wants to completely discontinue the RC with the widest-shortest order, it just needs to stop originating such PCBs from its AS. Since SCION PCBs have a limited lifespan, such PCBs are automatically removed from the control plane.

4.5 Reverse On-Demand RCs

In case of asymmetric inter-domain paths, which is the common case in the Internet, and in case of different criteria for forward and backward paths, on-demand RCs are not sufficient to achieve the optimal quality of service for both directions of communication as on-demand RCs only optimize paths from any AS to the originating AS, i.e, from clients to the server. To achieve optimal quality in both directions of communication, we introduce *Reverse On-Demand RCs*.

An AS initiates a new *Reverse On-Demand RC* with an on-demand SP to discover and optimize paths in the *reverse* direction of traditional routing protocols, i.e., *from itself to other ASes*. In a typical client-server setting, a Reverse On-Demand RC can be initiated from the server side to optimize the *return path from server to clients* according to the server's criteria. This enhances the quality of experience, as the server knows the best about the criteria of both the forward and backward paths.

The core novelty of this mechanism is the computation of paths in the same direction as data traffic typically flows. Unlike conventional routing protocols, where routing messages flow from the destination of the path toward the ASes that will use the path, i.e. constructing paths from any source to a specific destination, reverse on-demand RCs invert this flow. PCBs now propagate from the source of the path toward the potential destinations of the resulting paths, i.e. constructing paths from a specific source to all potential destinations.

Reverse on-demand RCs deviates from traditional beaconing model in SCION, requiring specific modifications to enable the correct propagation and handling of routing messages, i.e., PCBs, in the reverse direction. The following changes are necessary for reverse on-demand RCs.

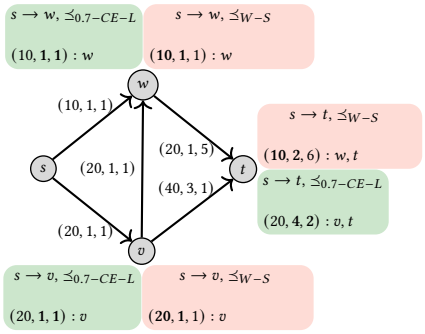


Figure 7: Example of reverse on-demand RCs.

- Every AS receiving such a PCB extends its path attribute in the direction of PCB traversal. This is in contrast to conventional routing, where the path attributes of the PCBs are extended in the opposite direction of PCB flow.
- Every AS receiving such a PCB, in addition to storing and advertising it if elected, executes the SP on all received PCBs for the specific OD-DC RC. Every AS sends one copy of the selected PCBs back to the source AS. This is because any AS is a destination to which the routes are being computed. The copy is sent back over the path described in the PCB. The source executes the SP on the received PCB copies to find the optimal path.
- The encoded path in the PCB is the optimized path from the source to a destination and should be used as is. In conventional routing, the optimized path is the reverse of what is encoded in the PCB and should be reversed.

Figure 7 shows an example of a reversed on-demand RCs.

Note that reverse RCs are only possible in a PAN architecture that benefits from stateless forwarding. This is because, in *stateful* forwarding (like traditional BGP with longest-prefix match), routers can only look up how to reach a specific destination in their forwarding tables; thus, routes can only be computed *to a specific destination*. Forwarding tables are not indexed by source, making it hard to install a reverse path, i.e. *from a specific source to any destination*.

4.6 Deployment Considerations

With parallel routing. ISPs deploy (new) SPs in a plug-and-play manner, giving them free choice in deploying SP. However, routing is a collaborative effort whose outcome greatly benefits from having as many ISPs as possible participating in RCs. To reconcile the latter and the former objective, we introduce a guideline for ISPs on the deployment of SPs. The principle behind this guideline is to prioritize primary network functions, which are crucial to all parties, e.g., connectivity, over the properties that are not crucial for the network function, e.g., computing environment-friendly paths. Therefore, we introduce three categories of SPs, which are organized in a pyramid depicted in Figure 8 whose bottom represents more critical SPs with less extensibility and opportunity for optimization, while the top represents the least critical SPs but with the highest extensibility and optimization opportunity. ISPs must prioritize the deployment of SPs closer to the bottom, and only deploy the higher levels if enough resources are available and such

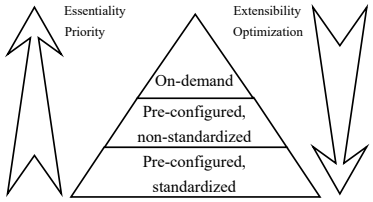


Figure 8: Three main categories of SP

deployment is beneficial to their business. These categories are as follows:

(1) At the base, there are SPs that must be standardized, i.e., the ISP cannot change the logic individually, and pre-configured by participating ISPs, i.e., end domains cannot influence path selection. These SPs are designed to enable the most basic network functions, e.g., connectivity. SPs to optimize paths for fundamental performance metrics, e.g., latency, can also be standardized and fall into this category.
(2) In the middle, there are SPs that are not standardized but are selected and pre-configured by each ISP, i.e., end domains cannot influence path selection. ISPs can choose to deploy these SPs according to their business and their analysis of their customers preferences. Note that such SPs can still be publicly published to be used by ISPs as plug-and-play programs without needing the ISPs to necessarily develop them in-house.
(3) At the top, there are *on-demand* SPs, a new way of route computation we propose in Sections 4.4 and 4.5, allowing end domains to influence route computation in ISPs on the path. End domains initiate a routing message containing their desired SP for optimizing paths. Upon receiving it, ISPs temporarily deploy the specified SP and execute it for all routing messages that pertain to the originator and include the same SP. These programs provide the most extensibility and opportunity for path optimization. They substitute standardization as all ISPs that accept them execute the very same program. However, in contrast to standardization, which can take years, on-demand SPs are instantly distributed.

IREC is an architecture that accommodates all three categories of SPs in a unified way using the eBPF technology [3], which is described in the next section.

5 Control Service Design

The control service of each SCION AS is responsible for inter-domain control-plane tasks, including beaconing (cf. Appendix E). This section describes a new design of the SCION control service that supports IREC routing mechanisms (cf. Section 4). To support IREC, an AS deploys this new control service. This new version is backward compatible and can be incrementally deployed.

5.1 Overview

The IREC-enabled control service has three main components: (1) the *ingress gateway*, receiving PCBs, storing them, and filtering

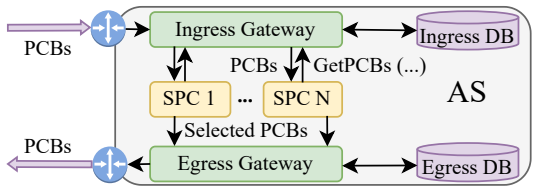


Figure 9: IREC-enabled control service for SCION.

them based on the AS-local policies, (2) *SP containers* (SPCs), executing SPs in isolation, and (3) the *egress gateway*, collecting the selected PCBs from the SPCs, filtering duplicate ones, and advertising them further to neighboring ASes. The egress gateway is also responsible for starting new RCs through originating PCBs. Figure 9 provides an overview of this design of the control service.

5.2 Ingress Gateway

When receiving a PCB from a neighboring AS, the ingress gateway verifies the included signatures and whether the path constructed by the PCB complies with the local AS' policies. The ingress gateway then stores the PCB in its *ingress database*. The ingress database stores PCBs indexed by their RC they belong to. The ingress gateway retrieves the SP from the originating AS if it has not yet cached the SP and caches it as long as there are PCBs specifying that SP. Note that there is no cyclic dependency in SP retrieval: The originating AS can always be reached via the path contained in one of the PCBs. The gateway periodically removes expired PCBs from the database.

5.3 SP Containers (SPCs)

A SPC is a sandboxed environment for executing the pluggable SPs. We design two types of SPCs: (1) static, executing one generic SP configured by the local AS, and (2) dynamic, which, in a round-robin way, executes various on-demand SPs and reverse on-demand SPs.

Instances of both SPC types request PCBs from the ingress gateway in a round-robin way. The ingress gateway serves requests of these two types of SPCs differently. A static SPC is provided with all received PCBs. A dynamic SPC is provided with the set of PCBs associated with an unprocessed $\texttt{ID}_{\texttt{RC}}$ together with the corresponding SP. An unprocessed $\texttt{ID}_{\texttt{RC}}$ is one with at least one new received PCB that is not processed by a dynamic SPC. After receiving the PCBs from the ingress gateway, a static SPC executes its statically configured SP. A dynamic SPC, on the other hand, executes the received SP per interface of the local AS since the received PCBs in each round all belong to the same $\texttt{ID}_{\texttt{RC}}$. Before every execution, a SPC excludes the PCBs that are not allowed to be propagated on the considered interface according to the AS-local policies. Upon execution, a SPC provides the SP with an interface through which the SP can request (1) the set of PCBs allowed to be propagated on the considered interface according to AS-local policies, and (2) available performance metric information about the intra-AS paths connecting all AS' interfaces to the considered interface, allowing for extending PCB attributes before selection (cf. Appendix B.2.2). The SP returns the set of selected PCBs for that execution. Since SPs defined by one ASes must be executed by other ASes, the programmatical interface between the SPC and the SP, which is used to retrieve the input PCBs, query available performance metrics and submit the selected results, must be standardized to ensure interoperability. Once all executions are performed, the SPC communicates the selected PCBs for each interface to the egress gateway. At this point, the system iterates and again requests new PCBs from the ingress gateway. The maximum number of SPs an AS can execute in parallel, therefore, corresponds to the number of SPCs in that AS.

5.4 Egress Gateway

PCB Origination. When creating new PCBs, the egress gateway of the originating AS adds the attribute of the inter-domain link from which the PCB is being emitted. To benefit from an on-demand or reverse on-demand RCs, the egress gateway adds an extension to the PCB with the URL of the respective SP and a flag specifying the reverse RC (cf. Appendix C). The egress gateway then signs the PCB and sends it to the ingress gateway of the neighboring AS connected to the considered interface.

PCB Propagation. When receiving selected PCBs for each interface from the SPCs, the egress gateway filters them by consulting its *egress database*. If a received PCB is not yet present in this database, the PCB is inserted together with the egress interface IDs for which it is selected. The egress gateway then performs the extension operation for each PCB selected for each interface: It appends an AS entry that, in addition to other necessary information, contains (1) the attributes of the intra-AS path connecting the interfaces where the PCB entered the AS and is going to exist the AS, and (2) the attributes of the inter-AS links connected to those two interfaces. The egress gateway sends the extended selected PCB over the selected inter-domain interface to the ingress gateway of the neighboring AS on the other end of the interface. If the `Rev` flag is set in the PCB, a copy of the PCB is also sent to its originating AS. If a received PCB is already present in the egress database, which can happen for PCBs selected by multiple SPCs or in different rounds of selection, the PCB is only propagated over those interfaces whose IDs are newly added to the database. Similarly to the ingress database, (soon-to-be) expired PCBs are removed from the egress database.

Path-segment Registration. To make the computed path segments available to endpoints, the egress gateway registers them at the path service of the AS. To allow usability, it tags each segment with the identifiers of the routing SPs by which it is computed.

6 Implementation and Evaluation

We first implement a proof-of-concept of IREC in a fork of the SCION open-source codebase [12] and then evaluate it through emulation using the Kathará network emulator [7, 17]. We further evaluate IREC's performance in terms of message complexity and convergence time using a simulation on a larger topology.

6.1 Implementation

We implement the control service proposed in Section 5 in the Go programming language as two separate applications bundled in two Docker [2] images: One application that implements the aggregated functionalities of ingress and egress gateways (GWs), and

another that implements SPCs. An instance of the SPC application can be configured to be a static or a dynamic SPC. SPCs communicate with gateways using remote procedure (gRPC) [5] calls. We use SQLite [13] for the ingress and egress databases. To ensure interoperability with neighboring ASes not supporting IREC, the ingress gateway listens on the same service address as the globally standardized service address of the legacy control service [11].

SPCs. To reduce maintenance effort and cost, we develop a unified implementation for static and dynamic SPCs. The type of SPC is configured upon initialization.

SPCs use eBPF [3] technology to execute SPs. This allows for (1) extensibility, as SPs can be written in various languages and compiled into eBPF bytecodes, and (2) safety, thanks to eBPF's built-in guarantees. Specifically, since eBPF is designed to run in the Linux kernel, a static verifier has been developed that performs a comprehensive analysis of the program before execution, checking for properties such as guaranteed termination, bounded loops and safe memory access. Programs that could potentially loop indefinitely, access invalid memory are rejected by the verifier. We use the user-space variant of eBPF, uBPF [14], as we do not want to run alien code from other ASes in the kernel. uBPF runs the eBPF bytecode within a virtual machine (VM) in the user space. As uBPF has limited helper functions compared to eBPF, we use the libxbgp [9] library, which enhances the uBPF VM with more helper functions and introduces a user-space verifier. Since uBPF is a library written in C and SPCs are written in Go, we define a C interface for calls to the library, export it, and invoke it from Go using Cgo [1]. As an alternative to eBPF, one may consider a runtime based on WebAssembly [15], which provides similar extensibility and isolation guarantees, but may only provide resource restrictions, and not termination guarantee.

A dynamic SPC creates a new VM for every execution of a SP and destroys it afterward. A static SPC creates the VM only once upon initialization. To enhance performance, we use the just-in-time compiler to translate eBPF bytecode to x86 instructions upon creating the VM. Before every execution, a SPC prepares the memory region for the SP. The memory region is used to provide PCBs to the SP as well as a working memory. PCBs are provided to the SP using FlatBuffers [4], allowing for fast access to variable-sized constructs without parsing or unpacking.

The SPC exposes helper functions with fixed identifiers that each SP must use to (1) retrieve the length and offset of the PCBs in memory, (2) query performance metrics and (3) submit selection results. Local performance metrics, such as latency on intra-AS and inter-AS links, are exposed via a key-value interface, using fixed semi-standardized keys that match the metadata an AS would include in a PCB. Metadata from other ASes is already available in the PCB extensions of the input PCBs.

We develop one SP in C, implementing the same selection operation as the one in the master branch of the SCION codebase at the time of writing this paper. The selection operation sorts paths based on their AS-hop lengths and selects the N-shortest paths with a configurable N[4]. The bytecode size of this SP is 12kB. The execution time of an algorithm depends on both the complexity of the SP and the number of input PCBs. Empirically, evaluations with varying input PCB sizes show that the SP completes in the order of milliseconds on modern hardware. More details are given in Appendix F. To prevent excessive resource usage, ASes can impose limits on the execution time of SPs. Given that most ASes operate multiple SPCs in parallel, this enables the execution of thousands of SPs per second.

[4]The SP selects $N-1$ shortest paths and one path that is most disjoint to the other $N-1$s. For simplicity, we say it selects the N-shortest paths.

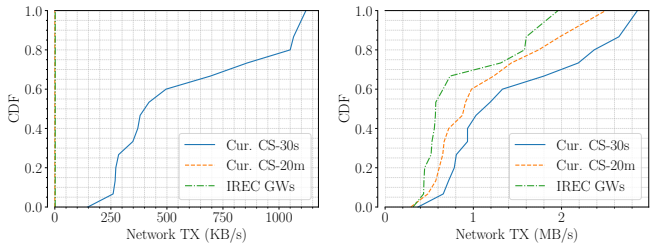
6.2 Evaluation

We evaluate our implementation by emulating a SCION network comprising 15 core ASes and a total of 581 inter-AS links. This topology is extracted from the CAIDA AS relationships with geographic annotations [21] data set, containing relationships and the locations of links between more than 12 000 ASes across the Internet. The extracted ASes are the ones with the highest degree in the data set. The distance between all pairs of extracted ASes, except one pair, is one. The distance of that single pair is two.

Due to the size of the topology, which cannot be emulated on a single machine, we use a variant of the Kathará emulator, Megalos [42], that performs network emulations in a Kubernetes [8] cluster. Megalos automates the creation of the cluster, deployment of containers of emulated network devices, and the connections between them. For our emulations, we instantiate 10 compute nodes in a data center of a cloud provider, all of the same instance type, with 8 vCPUs and 16 GB main memory. We carry out three experiments, each for one hour: (1) an IREC experiment in which all ASes deploy IREC with only one *dynamic* SPC and each AS starts one on-demand RC (cf. Section 4.4), (2) Current CS-30s, an experiment in which ASes deploy the current SCION control service (CS) with a beacon propagation interval of 30 seconds, and (3) Current CS-20m, an experiment with the current control service with a beacon propagation interval of 20 minutes. In all experiments, PCBs are originated only once at the beginning of the experiment, i.e. the origination interval is set longer than the experiment duration. In SCION experiments, we have a total of 83 virtual devices (containers), from which 53 are border routers, 15 are control services, one per AS, and another 15 are end hosts, one per AS. In the IREC experiment, there are 98 devices, where the 15 additional devices are the SPCs, one per AS. In all experiments, we disable the signature verification of PCBs as it is an expensive operation requiring more resources for emulations. In practice, the signature verification would not be a problem for the scalability of SPs, since the signature verification can be done outside the SP only once for all received PCBs. The verification requires the same per-PCB effort in the current control service as in IREC. The additional effort for signature verification would have been proportional to the number of PCBs received by the control service.

We sample CPU, memory, and bandwidth usage of every container every 5 s during each experiment and collect the metrics using Prometheus [10].

Bandwidth. Figure 10 illustrates the network transmit bandwidth use of selected containers in the three experiments. Figure 10a shows three distributions of the median bandwidth usages of (1) the current CS containers in the Current CS-30s experiment, (2) the current CS containers in the Current CS-20m experiment, and (3) the gateway containers (GWs) in the IREC experiment. Similarly,



(a) Median **(b) Peak**

Figure 10: Bandwidth usage of containers.

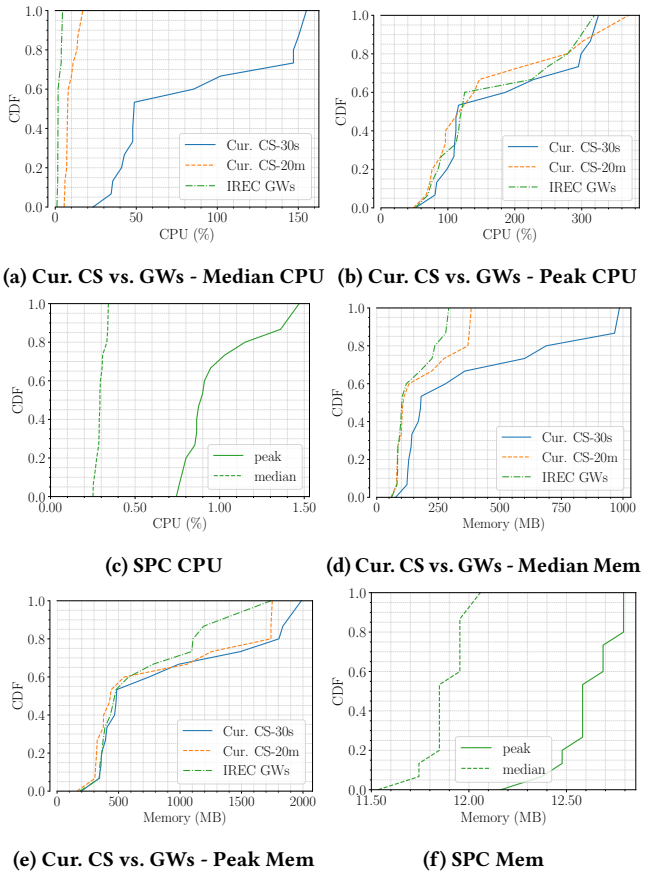
Figure 10b demonstrates the distributions of the peak bandwidth usages. The median and peak usages for each container are computed over the time series collected over the course of each experiment.

Among the three experiments, IREC and Current CS-20m both have a median TX rate of 0. This is because both do not propagate PCBs most of the time but for different reasons. Current CS-20m propagates PCBs only every 20 min, meaning that it does not propagate anything most of the time. IREC, on the other hand, keeps track of propagated PCBs in the egress gateway. PCBs selected by the SPC are not propagated if they have already been propagated (cf. Section 5.4). As the origination interval is longer than the duration of the experiment, there is only one origination of PCBs. Since the SP selects the same set of PCBs repeatedly, PCB propagation thus occurs only at the very beginning of the experiment. The peak network transmit rate, however, is similar in all experiments. This is because the peak rate represents the peak PCB propagation rate in all experiments, which is equal across experiments because the selection operation is the same across all of them.

CPU. Figure 11 illustrates the CPU, in the percentage of one CPU core, of selected containers in the three experiments. Figure 11a shows three distributions of the median CPU usages of (1) the current CS containers in the Current CS-30s experiment, (2) the current CS containers in the Current CS-20m experiment, and (3) the gateway containers (GWs) in the IREC experiment. Similarly, Figure 11b demonstrates the distributions of the peak CPU usages. Lastly, Figure 11c shows the distributions of median and peak CPU usages of the SPC containers in the IREC experiment. The median and peak usages for each container are computed over the time series collected over the course of each experiment.

Among the three experiments, IREC has the lowest median CPU usage of all because of a very low rate of PCB propagation as a result of filtering and not propagating duplicate PCBs at the egress gateway. Current CS-20m also has much lower CPU usage than Current CS-30s because of a much lower PCB propagation rate, i.e., 40× lower. The peak CPU usages, however, are similar in all experiments because of a similar peak PCB propagation rate.

The most important result is the very low CPU usage of SPCs, both at the peak and the median, compared to the current control service and gateways. This suggests the feasibility and scalability of SPs in uBPF VMs. This allows for running many SPCs in parallel with modest compute resources, allowing for the deployment of many SPs and the extensibility of routing criteria. Note that SPCs



(a) Cur. CS vs. GWs - Median CPU **(b) Cur. CS vs. GWs - Peak CPU**

(c) SPC CPU **(d) Cur. CS vs. GWs - Median Mem**

(e) Cur. CS vs. GWs - Peak Mem **(f) SPC Mem**

Figure 11: CPU and memory use of containers.

are always running and are executing SPs every 5 s even though they are filtered by the egress gateway.

Memory. Figure 11 illustrates the memory use, in MB, of selected containers in the three experiments. Figure 11d shows three distributions of the median memory usages of: (1) the current CS containers in the Current CS-30s experiment, (2) the current CS containers in the Current CS-20m experiment, and (3) the gateway containers (GWs) in the IREC experiment. Similarly, Figure 11e demonstrates the distributions of the peak memory usages. Lastly, Figure 11f shows the distributions of median and peak memory usages of the SPC containers in the IREC experiment. The median and peak usages for each container are computed over the time series collected over the course of each experiment.

We see similar trends for memory usage as we see for CPU usage, suggesting the superiority of IREC and the feasibility and scalability of SPCs (less than 13 MB peak memory use for any SPC), enabling the extensibility of routing criteria.

7 Large-Scale Simulations

In this section, we study the inter-domain behavior of parallel routing in a large network using simulations. By abstracting the implementations of IREC and the current control service, large-scale

simulations are significantly less resource-intensive than large-scale emulations (cf. Section 6.2) for the same large topology.

7.1 Simulation Setup and Topology

We simulate IREC in the ns-3-based [51] SCION simulator on the topology of the highest-degree 500 ASes extracted from the CAIDA AS relationships with geographic annotations [21] with more than 90 000 inter-AS links in the extracted topology. Using the locations of inter-AS links, we estimate the great circle propagation delay between border routers of each AS. We also annotate inter-AS links with bandwidths using degree gravity, which sets the link capacity proportionally to the product of the degrees of ASes at the two ends of the link. The bandwidth of a link is chosen from the set $\{1, 10, 25, 40, 100, 400\}$.

7.2 Experiments

We perform the following experiments:

Parallel RC Experiments

- All ASes advertise the *union* of optimal attributes according to *two* total orders $\preceq_{\alpha-CE-L}$, one with $\alpha = 1$ and the other with $\alpha = \frac{1}{2}$. Each total order selects the attribute with the lowest linear combination of carbon emission and latency, i.e., $\alpha \times CE + (1-\alpha) \times L$. We refer to this experiment as $\bigcup_{\alpha \in \{\frac{1}{2},1\}} \sqcap_{\preceq_{\alpha-CE-L}} A$, where A represents the set of attributes and $\sqcap_{\preceq_{\alpha-CE-L}} A$ represents the set of selected attributes from A according to order $\preceq_{\alpha-CE-L}$.
- Similar experiment but with three $\alpha \in \{\frac{1}{2}, \frac{2}{3}, 1\}$. We refer to this experiment as $\bigcup_{\alpha \in \{\frac{1}{2},\frac{2}{3},1\}} \sqcap_{\preceq_{\alpha-CE-L}} A$.

Single RC Experiments

- All ASes advertise the most preferred attributes on the intersection of the two total orders $\preceq_{\alpha-CE-L}$ with $\alpha \in \{\frac{1}{2}, 1\}$. We refer to this experiment as $\preceq \bigcap_{\alpha \in \{\frac{1}{2},1\}} \alpha-CE-L$.
- All ASes advertise the most preferred attributes on the intersection of the three total orders $\preceq_{\alpha-CE-L}$ with $\alpha \in \{\frac{1}{2}, \frac{2}{3}, 1\}$. We refer to this experiment as $\preceq \bigcap_{\alpha \in \{\frac{1}{2},\frac{2}{3},1\}} \alpha-CE-L$.
- All ASes advertise the most preferred attributes on the product order of the bandwidth and latency. In the product order, (w, l) is preferred to (w', l') if they are different and $w \geq w'$ and $l \leq l'$ [29]. We refer to this experiment as $\preceq_{W \times L}$.

In all experiments, there is only one PCB origination event at the beginning of the simulation. PCB selection and propagation at each AS happens periodically with the same period across all ASes. ASes are assumed to be perfectly time-synchronized. At every interval i, only PCBs received up to the previous interval $i-1$ are considered as candidates for the selection operation. In all experiments, if there are multiple paths corresponding to the selected attribute, only one of the paths is selected. To ensure determinism, with regard to selection and not arrival order, we use the following tie-breaker: (1) The path with the fewest hops is preferred; (2) Among paths with the same number of AS hops, the one whose sequence of AS numbers and interface identifiers is lexicographically the smallest is selected. This tie-breaker ensures determinism *only* in the *simulated* *periodic beaconing* explained above, where non-determinism in the arrival order of PCBs cannot impact path selection. In such a setup, it is guaranteed that selected paths with the same AS-hop length l are propagated and received by the next AS at the $l-1^{th}$ interval of the simulation (starting from 0) and are considered as candidates only at the next interval, i.e., l^{th} interval.

7.3 Results

Figure 12 compares the results of different experiments.

Convergence. Figure 12a shows the distribution of convergence times (in intervals) for all AS pairs, defined as the first interval after which no new PCBs are received. Despite a network diameter of 3, convergence takes 5–9 intervals depending on the experiment. Parallel RCs converge the fastest, with >95% within 7 intervals. Intersection-based routing converges >90% by interval 7. Product-order routing is the slowest: 10% by 7, 75% by 8, and 97% by 9 intervals—its full convergence requiring 9 intervals, compared to 7 (parallel) and 8 (intersection).

Selected Attributes at the Stable State. Figure 12b presents the distribution of the number of attributes selected by each AS from each of its interfaces toward any destination AS after convergence, across over 95 million data points. In all scenarios, over 70% of interface–AS pairs select a single attribute, indicating consistent choices across parallel RCs. Since parallel RCs take the union of individually selected attributes, the maximum number of selected attributes is bounded by the number of parallel RCs (two or three). For intersection-based routing, 99% of cases select fewer than six attributes, though the count can reach up to 20 in rare cases. Product-order routing shows a similar distribution but with a noticeably shorter tail, capped at six.

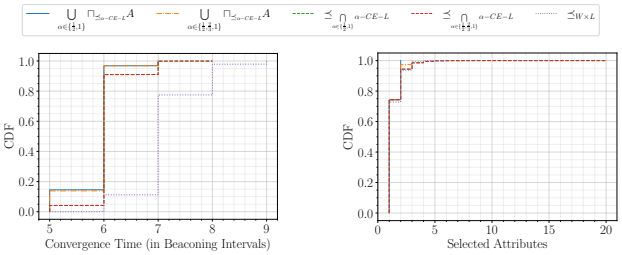
Message Complexity. Figure 12c Shows PCB count distributions per inter-domain interface across > 192,000 points (from 96,000 links). Message complexity of different experiments compares as follows: parallel routing (2 linear combinations) < parallel routing (3 linear combinations) < intersection-based routing < bandwidth-latency product-order, with the latter having the highest complexity and longest tail (>12,500 PCBs/interface), over twice that of parallel routing with 3 linear combinations.

8 Discussion

End-to-End Performance. RCs optimize paths at the interface group level, but end-to-end optimality requires clients to know the correct interface group for reaching a server—information not included in PCBs to reduce overhead. For metrics like latency or bandwidth, clients can probe across interface groups to find optimal paths, though this introduces delay. Future work may augment DNS responses with metric hints for interface group selection.

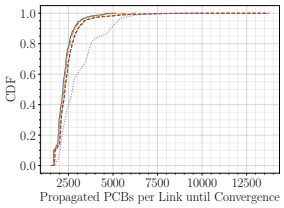
Executing Alien Code.

Executing code from other ASes in on-demand RCs raises security concerns. However, authenticated PCBs, our system design, and best practices mitigate major threats. Malicious SPs may aim to access confidential data or launch denial-of-service (DoS) attacks by exhausting SPC resources. First, SCION's signed PCBs reveal the identity of misbehaving ASes, discouraging abuse due to reputational risk. Second, IREC's architecture (Section 5) ensures complete isolation of SPCs, so even if one is overloaded, others



(a) Convergence time per AS pair.

(b) Selected attributes per source-destination pair at the stable state.



(c) Total propagated PCBs on each interface.

Figure 12: Simulation results

remain unaffected. Operators can limit resource usage and evict long-running SPs; the eBPF verifier guarantees termination. To ensure basic connectivity, each AS must run a static SPC with a generic SP. Third, SPs execute as uBPF bytecode within isolated VMs, providing two layers of isolation. uBPF code accesses only local memory, cannot interact with the network stack, and returns results to its SPC. As a best practice, sensitive data should not be stored on SPC hosts.

Ensuring that ASes Execute (Reverse) On-Demand SPs. On-demand and reverse on-demand RCs are only useful if ASes are committed to executing their associated SPs. We argue that ASes willingly execute the programs as it is to their best benefit; thus, there is no need for an enforcement mechanism. The reason is that end ASes specifying such programs want to later use the optimal paths computed by those programs for sending traffic. Therefore, if an AS on the optimal path executes the program, the traffic goes through that AS, increasing its revenue. Thus, they have a strong incentive to execute the programs. If an AS can somehow realize that it is not on the optimal path for a program, it probably does not execute it. In this case, there is no harm in not executing the program because even if it were executed, the resulting path would not have been selected by endpoints. It is, however, highly unlikely that an AS can realize whether or not it is on an optimal path for a SP because (1) an AS only sees the path attributes up to its AS and not further downstream, and (2) a non-curious AS only sees the bytecode of the SP and not the source code; thus, they do not see the selection criteria unless they perform analysis on the bytecode.

9 Related Work

Extensible Inter-Domain Routing. XIA [40], Trotsky [38], and xBGP [53] advance Internet extensibility. XIA supports evolving principals; Trotsky enables backward-compatible architectural changes; xBGP allows BGP extensions via user-defined eBPF bytecode, letting operators deploy features without vendor support or standardization delays. However, xBGP lacks support for multi-criteria optimization and criteria defined by end domains.

Multi-Criteria Path Optimization. Multi-objective routing models path metrics as Cartesian products ordered via product orders [18, 28, 37], but cover limited criteria. Sobrinho et al. [47] ensure optimality on multiple routing criteria by advertising the set of most preferred paths on the intersection of criteria. However, they do not count for extensibility of criteria.

On-Demand Routing. Yampolskiy et al. [54] and Route Bazaar[19] enable QoS-aware routing via endpoint requests and AS agreements. In contrast, IREC is a general-purpose architecture decoupled from bilateral QoS negotiations and targets optimal path computation rather than constraint satisfaction—though it can support constraints via (reverse) on-demand RCs.

10 Conclusion

Despite tremendous advancements witnessed in the realm of information technology over the past decades, along with a continuous increase in dependence on communication networks, inter-domain routing has endured a conspicuous lack of transformation over the course of the past 25 years. In particular, despite the ever-increasing diversity of applications' communication requirements, BGP has largely remained static over the past 25 years.

IREC overcomes those limitations by enabling the extension of routing optimality criteria. Taking advantage of the opportunities provided by the SCION path-aware Internet architecture, IREC introduces parallel RCs, laying the foundation for the extension of path optimization criteria. Building on parallel RCs, we introduce on-demand and reverse on-demand RCs, enabling both end ASes of traffic to express their path optimality criteria to the control plane without waiting for standardization, vendor implementation, or adoption by operators.

By emulating our uBPF-based implementation in the SCION codebase for a realistic topology, we showed the practicality of the introduced mechanisms. The ns-3 simulations on large realistic topologies demonstrate that parallel RCs, i.e., the building block of IREC, require less effort in terms of message complexity, than a single RC selecting most preferred attributes on the intersection of the same criteria.

IREC opens up new opportunities for exciting research in inter-domain routing, with the potential to enhance communication quality for endpoints and applications.

Acknowledgment

We would like to thank the anonymous reviewers and our shepherd Panda Aurojit for their valuable feedback. We gratefully acknowledge support from ETH Zurich, and from the Zurich Information Security and Privacy Center (ZISC). The work of J. L. Sobrinho is supported by FCT/MECI through national funds and, when applicable, EU funds under UID/50008: Instituto de Telecomunicações.

References

[1] [n. d.]. Cgo Documentation. https://pkg.go.dev/cmd/cgo archived at https://perma.cc/5N5A-2UND.

[2] [n. d.]. docker. https://www.docker.com/ archived at https://perma.cc/4APP-FENR.

[3] [n. d.]. eBPF-Dynamically program the kernel for efficient networking, observability, tracing, and security. https://ebpf.io archived at https://perma.cc/7TSN-89WR.

[4] [n. d.]. FlatBuffers: Memory Efficient Serialization Library. https://flatbuffers.dev/ archived at https://perma.cc/MC9D-DNVQ.

[5] [n. d.]. gRPC. https://grpc.io/ archived at https://perma.cc/G8C5-VBCT.

[6] [n. d.]. Juniper vs Cisco BGP Med Attribute. https://skminhaj.wordpress.com/2014/12/23/juniper-vs-cisco-bgp-med-attribute/ archived at https://perma.cc/4STV-P6DW.

[7] [n. d.]. Kathará. https://www.kathara.org/index.html archived at https://perma.cc/KUE4-Q6AA.

[8] [n. d.]. Kubernetes. https://kubernetes.io/ archived at https://perma.cc/9WXX-5B5N.

[9] [n. d.]. libxBGP. https://github.com/pluginized-protocols/libxbgp archived at https://perma.cc/9F8N-BB4D.

[10] [n. d.]. Prometheus. https://prometheus.io/ archive at https://perma.cc/3JA6-MTGY.

[11] [n. d.]. SCION Control Service Documentation. https://docs.scion.org/en/latest/manuals/control.html archived at https://perma.cc/M8ZB-U4AD.

[12] [n. d.]. scionproto. https://github.com/scionproto/scion.

[13] [n. d.]. SQLite. https://www.sqlite.org/index.html archived at https://perma.cc/34PT-R3EC.

[14] [n. d.]. Userspace eBPF VM - Github.com. https://github.com/iovisor/ubpf archived at https://perma.cc/6TVX-7AA3.

[15] [n. d.]. WebAssembly. https://webassembly.org archived at https://perma.cc/DWT5-EMJW.

[16] Henry Birge-Lee, Sophia Yoo, Benjamin Herber, Jennifer Rexford, and Maria Apostolaki. 2024. TANGO: Secure Collaborative Route Control across the Public Internet. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[17] Gaetano Bonofiglio, Veronica Iovinella, Gabriele Lospoto, and Giuseppe Di Battista. 2018. Kathará: A container-based framework for implementing network function virtualization and software defined networks. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*.

[18] James Brumbaugh-Smith and Douglas Shier. 1989. An Empirical Investigation of some Bicriterion Shortest Path Algorithms. *European Journal of Operational Research* (1989).

[19] Ignacio Castro, Aurojit Panda, Barath Raghavan, Scott Shenker, and Sergey Gorinsky. 2015. Route Bazaar: Automatic Interdomain Contract Negotiation. In *Proceedings of the USENIX Conference on Hot Topics in Operating Systems (HOTOS)*.

[20] Center for Applied Internet Data Analysis (CAIDA). [n. d.]. AS-Relationships. https://www.caida.org/data/as-relationships/ archived at https://perma.cc/RYN2-BSNB.

[21] Center for Applied Internet Data Analysis (CAIDA). [n. d.]. AS Relationships – with Geographic Annotations. https://www.caida.org/data/as-relationships-geo/ archived at https://perma.cc/S4XX-Y2EK.

[22] Laurent Chuat, Markus Legner, David Basin, David Hausheer, Samuel Hitz, Peter Müller, and Adrian Perrig. 2022. *The Complete Guide to SCION.* Springer.

[23] cyberlink. [n. d.]. SCION-Internet - Hochsicherer Datenaustausch. https://www.cyberlink.ch/scion archived at https://perma.cc/G7XC-ZBKK.

[24] Bin Da and Marco Carugi. 2020. *Representative use cases and key network requirements for Network 2030.* Technical Report. ITU-T.

[25] Igor Ganichev, Bin Dai, P. Brighten Godfrey, and Scott Shenker. 2010. YAMR: Yet Another Multipath Routing Protocol. *ACM SIGCOMM Computer Communication Review (CCR)* (2010).

[26] GÉANT. [n. d.]. Infoshare: SCION Access for Universities and Research Institutes. https://connect.geant.org/2022/11/18/infoshare-scion-access-for-universities-and-research-institutes-24-nov-2022 archived at https://perma.cc/PE4K-S36K.

[27] K. Oanh Ha. [n. d.]. When Your Boss Becomes a Hologram. https://www.bloomberg.com/news/articles/2022-03-03/big-tech-and-startups-look-to-3d-hologram-for-travel-free-communication archived at https://perma.cc/5KXV-LMUX.

[28] Pierre Hansen. 1980. Bicriterion Path Problems. In *Proceedings of the Multiple Criteria Decision Making Theory and Application.*

[29] Egbert Harzheim. 2005. *Ordered Sets.* Springer.

[30] Intercloud. [n. d.]. Securing cloud connectivity with SCION. https://www.intercloud.com/our-resources/blog/securing-cloud-connectivity-with-scion archived at https://perma.cc/CYX4-2H25.

[31] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: Experience with a Globally-Deployed Software Defined Wan. In *Proceedings of the ACM SIGCOMM Conference*.

[32] Karrierone. [n. d.]. Building the future of connectivity. https://scion.karrier.one archived at https://perma.cc/UH8S-DBZM.

[33] Cyrill Krähenbühl, Seyedali Tabaeiaghdaei, Christelle Gloor, Jonghoon Kwon, Adrian Perrig, David Hausheer, and Dominik Roos. 2021. Deployment and Scalability of an Inter-Domain Multi-Path Routing Infrastructure. In *Proceedings of the ACM SIGCOMM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*.

[34] Nate Kushman, Srikanth Kandula, Dina Katabi, and Bruce M. Maggs. 2007. R-BGP: Staying Connected In a Connected World. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[35] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. 2001. Delayed Internet routing convergence. *IEEE/ACM Transactions on Networking* (2001).

[36] Yong Liao, Lixin Gao, Roch Guerin, and Zhi-Li Zhang. 2008. Reliable Interdomain Routing through Multiple Complementary Routing Processes. In *Proceedings of the ACM SIGCOMM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*.

[37] Ernesto Queirós Vieira Martins. 1984. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research* (1984).

[38] James McCauley, Yotam Harchol, Aurojit Panda, Barath Raghavan, and Scott Shenker. 2019. Enabling a Permanent Revolution in Internet Architecture. In *Proceedings of the ACM SIGCOMM Conference*.

[39] Murtaza Motiwala, Megan Elmore, Nick Feamster, and Santosh Vempala. 2008. Path Splicing. In *Proceedings of the ACM SIGCOMM Conference*.

[40] David Naylor, Matthew K. Mukerjee, Patrick Agyapong, Robert Grandl, Ruogu Kang, Michel Machado, Stephanie Brown, Cody Doucette, Hsu-Chun Hsiao, Dongsu Han, Tiffany Hyun-Jin Kim, Hyeontaek Lim, Carol Ovon, Dong Zhou, Soo Bum Lee, Yue-Hsun Lin, Colleen Stuart, Daniel Barrett, Aditya Akella, David Andersen, John Byers, Laura Dabbish, Michael Kaminsky, Sara Kiesler, Jon Peha, Adrian Perrig, Srinivasan Seshan, Marvin Sirbu, and Peter Steenkiste. 2014. XIA: Architecting a More Trustworthy and Evolvable Internet. *SIGCOMM Computer Communication Reviews (CCR)* (2014).

[41] Donghong Qin, Jiahai Yang, Zhuolin Liu, Jessie Wang, Bin Zhang, and Wei Zhang. 2012. AMIR: Another Multipath Interdomain Routing. In *Proceedings of the IEEE International Conference on Advanced Information Networking and Applications (AINA)*.

[42] Mariano Scazzariello, Lorenzo Ariemma, Giuseppe Di Battista, and Maurizio Patrignani. 2021. Megalos: A Scalable Architecture for the Virtualization of Large Network Scenarios. *Future Internet* (2021).

[43] Simon Scherrer, Markus Legner, Adrian Perrig, and Stefan Schmid. 2021. Enabling Novel Interconnection Agreements with Path-Aware Networking Architectures. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 116–128. doi:10.1109/DSN48987.2021.00027

[44] Simon Scherrer, Seyedali Tabaeiaghdaei, and Adrian Perrig. 2023. Quality Competition Among Internet Service Providers. In *Proceedings of the IFIP International Symposium on Computer Performance, Modeling, Measurements and Evaluation (PERFORMANCE)*.

[45] João Luís Sobrinho. 2002. Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet. *IEEE/ACM Transactions on Networking* (2002).

[46] João Luís Sobrinho. 2005. An algebraic theory of dynamic network routing. *IEEE/ACM Transactions on Networking* (2005).

[47] João Luís Sobrinho and Miguel Alves Ferreira. 2020. Routing on Multiple Optimality Criteria. In *Proceedings of the ACM SIGCOMM Conference*.

[48] Swisscom. [n. d.]. SCION – for secure data transmission. https://www.swisscom.ch/en/business/enterprise/offer/wireline/scion.html archived at https://perma.cc/8MDB-62PS.

[49] Seyedali Tabaeiaghdaei, Simon Scherrer, Jonghoon Kwon, and Adrian Perrig. 2023. Carbon-Aware Global Routing in Path-Aware Networks. In *Proceedings of ACM International Conference on Future Energy Systems (e-Energy)*.

[50] Telindus. [n. d.]. SCiON A New Architecture that Overcomes the Limitations of the Internet. https://www.proximusnxt.lu/en/scion archived at https://perma.cc/5QQK-J2GP.

[51] University of Washington NS-3 Consortium. [n. d.]. ns-3 Network Simulator. https://www.nsnam.org/ archived at https://perma.cc/4S4N-8VGV.

[52] Feng Wang and Lixin Gao. 2009. Path Diversity Aware Interdomain Routing. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*.

[53] Thomas Wirtgen, Tom Rousseaux, Quentin De Coninck, Nicolas Rybowski, Randy Bush, Laurent Vanbever, Axel Legay, and Olivier Bonaventure. 2023. xBGP: Faster Innovation in Routing Protocols. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[54] Mark Yampolskiy, Wolfgang Hommel, Bernhard Lichtinger, Wolfgang Fritz, and Matthias K. Hamm. 2010. Multi-domain End-to-End (E2E) Routing with Multiple QoS Parameters – Considering Real World User Requirements and Service Provider Constraints. In *Proceedings of the IARIA International Conference on*

Evolving Internet (INTERNET).
[55] Xiaowei Yang, David Clark, and Arthur W. Berger. 2007. NIRA: A New Inter-Domain Routing Architecture. *IEEE/ACM Transactions on Networking* (2007).
[56] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, Victor Lin, Colin Rice, Brian Rogan, Arjun Singh, Bert Tanaka, Manish Verma, Puneet Sood, Mukarram Tariq, Matt Tierney, Dzevad Trumic, Vytautas Valancius, Calvin Ying, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2017. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *Proceedings of the ACM SIGCOMM Conference*.
[57] Xia Yin, Dan Wu, Zhiliang Wang, Xingang Shi, and Jianping Wu. 2015. DIMR. *Computer Networks* (2015).
[58] Ming Zhu, Dan Li, Ying Liu, Dan Pei, KK Ramakrishnan, Lili Liu, and Jianping Wu. 2015. MIFO: Multi-path Interdomain Forwarding. In *Proceedings of the ACM International Conference on Parallel Processing (ICPP)*.

A Considerations Regarding Hierarchical Routing of SCION

A.1 IREC Only for Core Beaconing

Given the two levels of routing hierarchy in SCION, i.e., core and intra-ISD beaconing, we argue that IREC is essential to the core beaconing and is unnecessary for the intra-ISD beaconing.

In the core of the network, the number of core-path segments is super-exponentially large because of the dense interconnection of ASes and the possibility of valley-full routing. In such circumstances, discovering all possible core-path segments is practically impossible due to the overwhelming communication and computation costs. Therefore, to discover core-path segments that are optimal according to specific criteria, it is necessary to take the optimality criteria into account for core-segment computation during core beaconing.

In contrast, sparse intra-ISD topologies, together with the unidirectional intra-ISD beaconing from core to leaf ASes that builds a directed acyclic graph (DAG), make it possible to discover *all* possible up- and down-path segments with negligible cost. This is empirically shown by Krähenbühl et al. [33]. Hence, taking into account optimality criteria for intra-ISD beaconing does not provide any benefit and is therefore unnecessary.

B Incorporating Intra-AS Topologies for Optimality

Every AS on an inter-domain path is a network of routers distributed across different locations. Consequently, an inter-domain path is a sequence of *intra-AS* paths connected by inter-domain links. Therefore, the attributes of intra-AS paths contribute to the attributes of inter-domain paths. However, the SCION control plane abstracts away the internal topology of ASes by modeling them as abstract nodes. This means that the attributes of *intra-AS* paths within ASes are not taken into account in computing the attributes of inter-domain paths, potentially resulting in the selection of suboptimal inter-domain paths. In the following two sections, we elaborate on how this modeling can cause suboptimality and how we address these problems.

B.1 Last-Mile Path Matters

B.1.1 Problem. Modeling ASes as abstract nodes ignores the last-mile intra-AS path from the last border router to the traffic destination within the end AS. Such modeling falsely assumes that any inter-domain path that is optimal for reaching a border router of an AS is also optimal for reaching any endpoint within that AS. SCION beaconing, until the time of writing this paper, works based on this assumption: It computes routes on a per-AS basis, i.e., it selects optimal paths to reach *any entry point* of an AS, which can result in suboptimal paths to endpoints.

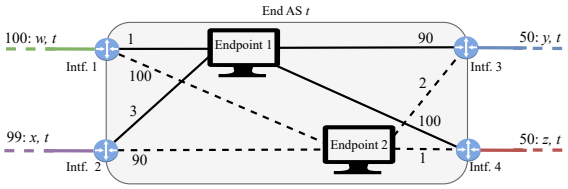


Figure 13: An example of per-AS RC resulting in suboptimal end-to-end paths. Each line represents a path. Lines within the AS represent intra-AS paths to endpoints, and lines out of the AS represent inter-domain paths to AS t received by an AS v (not shown). While the shortest path to endpoint 1 enters AS t at interface 1, a per-AS RC according to the shortest order $\preceq_S$ results in selecting the two paths reaching interfaces 3 and 4.

Figure 13 illustrates an example of per-AS RC resulting in suboptimal paths to endpoints. A per-AS RC according to the shortest order $\preceq_S$ selects the shortest inter-domain paths to AS t, i.e., the ones with length 50 to interfaces 3 and 4. Using these paths to reach endpoint 1 results in data packets being forwarded on intra-AS paths with lengths 90 or 100 from interfaces 3 or 4 to endpoint 1. This results in a total length of 140 or 150 to endpoint 1. On the other hand, selecting and using the inter-domain path to interface 1 results in a total length of 101 to endpoint 1, i.e., the shortest possible length to that endpoint. However, the per-AS RC never selects the path to interface 1, resulting in suboptimal end-to-end performance.

For core-path segments combined with up- or down-path segments, the endpoint in the above example is the border router where the up or down segment ends or starts, respectively.

B.1.2 Solution. To better account for the last-mile intra-AS paths to endpoints, we propose computing routes pertaining to groups of interfaces of each AS. This means selecting and advertising optimal attributes to reach every group of AS interfaces. This approach is equivalent to considering every group of interfaces as a different AS to which routes are computed without creating new ASes.

The intuition behind this approach is that the interfaces of an AS can be grouped together based on the attributes of the paths connecting them to different services hosted in the AS. Interfaces in the same group offer similar performance, with regard to a subset of metrics, to reach all services hosted in the AS. Therefore, this approach can provide benefits similar to announcing different prefixes from different subsets of routers with BGP in today's Internet.

RC pertaining to interface groups requires the following actions:
ASes Starting RCs (End ASes)

- *Defining interface groups and assigning an identifier to them.* Different ways of grouping can co-exist according to different performance metrics. For example, location-based

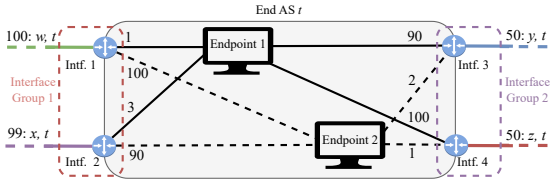


Figure 14: An example of interface groups enhancing the optimality of paths to endpoint 1 with regard to shortest order $\preceq_S$. The shortest path to interface group 1 is 99: x, t to interface 2, resulting in a length of 102 to endpoint 1. Without interface groups, the shortest paths to AS t are 50: y, t and 50: z, t, resulting in a length of 140 to endpoint 1.

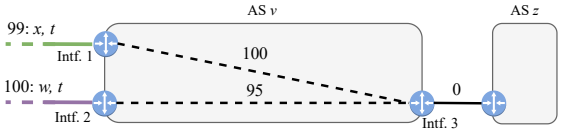


Figure 15: An example of how abstracting internal AS topology results in advertising suboptimal attributes. Attribute b = 99: x, t is preferred to attribute c = 100: w, t according to the shortest order $\preceq_S$. By abstracting the internal topology of AS v, attribute b is chosen to be advertised to AS z. This results in a length of 199, while selecting c would result in a length of 195.

groups and bandwidth-based groups can co-exist for latency and bandwidth optimizations, respectively. Therefore, the interface groups are not mutually exclusive. An interface in the central Europe location-based group can be part of the high-bandwidth group together with other high-bandwidth interfaces around the world.

- *Starting at least one RC for every interface group.* This is done by originating PCBs, each containing the interface group identifier, from all interfaces that are part of that interface group. The inclusion of an interface-group identifier is independent of whether the RC is generic, on-demand, or reverse on-demand, depending on whether the AS includes a SP in the PCBs as well. Multiple RCs with different SPs can be started for the same interface group.

Note that an AS can modify its interface groups. In that case, it needs to start new RCs accordingly.

ASes Receiving PCBs with Interface-Group Identifiers

- *Performing the selection operation separately on PCBs pertaining to every interface group of every AS.* In doing so, they group PCBs based on their interface group identifier (and ISD-AS number, to avoid collision of interface group identifiers in different ASes).
- *Advertising selected PCBs pertaining to every interface group to neighboring ASes.*

Figure 14 shows how interface groups can be defined in the same topology as Figure 13 to enhance the optimality of paths to endpoint 1 with regard to shortest order $\preceq_S$. AS t defines two interface groups: group 1, including interfaces 1 and 2, and group 2, including interfaces 3 and 4. AS v (not shown in the figure) receives paths starting from these four interfaces and groups these paths according to their interface groups: Two paths pertaining to group 1, i.e., 99: x, t and 100: w, t, and two paths pertaining to group 2, i.e., 50: y, t and 50: z, t. The selection operation is performed for every interface group separately: Path 99: x, t is elected for reaching interface group 1 of AS t, and both 50: y, t and 50: z, t are selected to reach interface group 2 of AS t. Using 99: x, t to reach endpoint 1 results in a near-optimal length to the endpoint, i.e., 102, while a per-AS path computation only selects 50: y, t and 50: z, t, resulting in at least a length of 140 to endpoint 1.

Note that the selected path to interface group 1, i.e., 99: x, t, still does not result in the optimal length to endpoint 1. The optimal length to endpoint 1, i.e., 101, only results from using 100: w, t, which is not selected as it is longer than 99: x, t to the same interface group 1. To address this issue, AS t can define more granular interface groups by creating an interface group that only contains interface 1 and another interface group that only contains interface 2. In that case, 100: w, t, is also selected, resulting in optimal length to endpoint 1.

The remaining research question is how to realize that endpoint 1 should be reached over interface group 1. Unless this question is answered, the proposed mechanism cannot provide a significant benefit. This problem is also due to per-AS instead of per-prefix routing in SCION. One way to solve this problem is to use DNS to provide information on the optimal interface groups to reach a service. However, providing a concrete solution to this problem is out of the scope of this paper.

B.2 A False Notion of Non-Isotonicity

B.2.1 Problem. Modeling ASes as abstract nodes can result in advertising suboptimal paths by selecting attributes that do not remain optimal after extension by intra-AS attributes. Figure 15 provides an example of this phenomenon for a RC according to the shortest order $\preceq_S$, selecting the attribute with the shortest length. There are two received attributes, i.e., b = 99: x, t and c = 100: w, t. Clearly, $b \preceq_S c$ because of the shorter length. By abstracting the internal topology of AS v, attribute b is chosen to be advertised to AS z. However, $a[zv] \oplus a[Intf.3, Intf.2] \oplus c \preceq_S a[zv] \oplus a[Intf.3, Intf.1] \oplus b$, because $a[zv] \oplus a[Intf.3, Intf.2] \oplus c = 195 : v, w, t$, while $a[zv] \oplus a[Intf.3, Intf.1] \oplus b = 199 : v, x, t$. In other words, the relative preference between b and c is not preserved when they are advertised to the next AS despite the fact that $\preceq_S$ is isotone. This is because the attributes of the two paths from interface 3 to t are in fact not b and c but are $a[Intf.3, Intf.1] \oplus b$ and $a[Intf.3, Intf.2] \oplus c$.

Note that this is not a problem in routing protocols in which every router computes routes from itself to the destination: Routes received from a peer router in the same AS are already extended by the attributes of the intra-AS path that connects them. Therefore, the differences in intra-AS paths are visible in inter-domain paths. In SCION, however, it is the logically centralized control service of

each AS that participates in inter-domain routing, not the border routers.

B.2.2 Solution. To address this problem, we propose that the control service of each AS selects and extends routes from the perspective of every border router such that it computes the same set of paths that the routers themselves would select. Therefore, for every RC, the control service performs the following steps *for every interface i* in the set of interfaces I:

(1) Extend every inter-domain attribute pertaining to the RC that is received from any interface o other than i by $a[io]$, i.e., the attribute of the intra-AS path connecting interfaces i and o. This step builds set $E_i := \{a[io] \oplus c | c \in C[o], o \in I \setminus i\}$, where $C[o]$ represents inter-domain attributes pertaining to the RC that are received from interface o.
(2) Perform selection operation on E_i.

In Figure 15, when selection is performed on $E_3 = \{a[Intf.\,3, Intf.\,1] \oplus b = 199 : v, x, t, a[Intf.\,3, Intf.\,2] \oplus c = 195 : v, w, t\}$, the optimal attribute $a[Intf.\,3, Intf.\,2] \oplus c = 195 : v, w, t$, is selected.

C PCB Extension

We introduce the IREC PCB extension that provides information to ASes participating in IREC. Each PCB can have at most one IREC extension included in its first AS entry. Only the AS starting a RC, i.e., the AS originating the PCB, can include this extension. ASes receiving such a PCB must not modify this extension. ASes not supporting IREC must ignore the extension, meaning that they must *neither* discard the PCB *nor* remove the extension. This ensures connectivity and allows for partial optimization by IREC-enabled ASes.

This extension contains the following fields:

- SP Identifier (`SP`): Specifying the URL using which the SP can be retrieved from the originating AS. This field is used for on-demand and reverse on-demand RCs (cf. Sections 4.4 and 4.5). If not specified, the PCB is considered for generic RCs.
- Reverse Computation Flag (`Rev`): This flag determines whether the PCB belongs to a reverse on-demand RC (cf. Section 4.5). This flag can only be set if the `SP` is not null. If set, an AS extends attributes in reverse and returns a copy of the PCB to the originating AS.
- Interface Group Identifier (`IG`): An integer specifying the interface group for which the RC computes the routes (cf. Appendix B.1.2). This field can be set independently of the other two fields.

A RC can be uniquely identified by (Originating ISD, Originating AS, `SP`, `IG`, `Rev`). We call this tuple the *RC Identifier* ($\texttt{ID}_{\texttt{RC}}$).

Note that IREC can also work on PCBs without such an extension. PCBs without the extension are considered for all parallel generic RCs. This is because generic RCs are oblivious to `SP` and `Rev` fields, and `IG` is optional.

D Case Study: Optimizing Paths for Client-Server Applications

We study how the IREC architecture can be used to enhance the quality of experience for client-server applications by allowing optimization of SCION core-path segments for the criteria desired by each application. In this study, we assume a common case in modern applications, where the client application is developed by the same entity that develops and operates the server side. For example, the YouTube app is developed by Google, and the content is served by Google-developed servers on the cloud run by Google. In such cases, the entity (e.g., Google) knows which path optimality criteria suit either direction of the communication (i.e., client-to-server and server-to-client) the best. This also applies to tenants of cloud providers.

Which Side Should Start the RCs?. We argue that the server side of such applications should leverage the mechanisms provided by IREC to optimize paths in both directions between clients and servers. The reasons are fourfold:

- It is significantly more scalable to start one RC from the server side (or a handful if the server is in the customer cones of multiple core ASes) than to start thousands of computations from all core ASes with clients in their customer cones.
- A cloud provider that hosts a service can operate core AS(es) and have full control over its core-segment computations or have an influence on its providers' core-segment computations. This control can be offered as a service to cloud tenants. On the other hand, client applications are not expected to have a significant, if any, influence on which RCs their ISP starts.
- Starting RCs from the client side would require either the involvement of end users or the standardization and development of an interface between client applications and ISPs to automate the expression of the SP to ISPs. None of these are required on the server side. The cloud provider offers a user interface (UI) or an infrastructure as code (IaC) model to tenants, using which they can define their SPs.
- Changing SPs on the server side is dramatically easier, in terms of operational effort, than on the client side. On the server side, the defined SP on the cloud UI or IaC needs to be updated. On the client side, the application developer would need to release a patch for (up to) billions of client applications.

Required RCs. To optimize core-path segments between clients and a server of an application, the core AS whose customer cone includes the server's AS needs to (1) start an on-demand optimization that computes paths from clients to the server according to the criteria for reaching the server, and (2) start a reverse on-demand optimization that computes paths from the server to clients according to the criteria for reaching the clients. Note that the SPs of these RCs can use different orders. For example, for a streaming service, a high-bandwidth path may be required from the server to the clients, while a low-latency path may be required for acknowledgment packets from the clients to the server.

Path-segment Lookup. Once these core segments are computed, they need to be looked up by the endpoints. We argue that the segments for both directions of the communication should be looked up on the client side. This is because path lookup is an expensive operation and can overwhelm a server for a large number of clients. Furthermore, the core AS on the client side receives and stores optimal segments for both directions of the communication. On the contrary, the core AS on the server side receives the optimal paths only for the server-to-client direction of communication through a reverse on-demand RCs.

IREC-aware Core Segment Selection at Endpoints. The core-segment lookup operation can result in multiple segments for each direction of communication. If they are computed by a single RC, a random one can be returned to the client. However, it can be the case that the segments are computed by different RCs, and each is optimal for different sets of criteria. In such cases, the client application should know which RC is tailored to its communication criteria. To solve this problem, we propose the IREC-aware core-path segment selection procedure. This means that endpoints choose the segments that are computed by their desired RC instead of performing segment optimization themselves. This saves endpoints from repeating the same computation already performed by RCs. In addition, it significantly reduces the number of segments returned to clients from the path service, enhancing the scalability of segment lookup. Most importantly, client applications remain much simpler, as changing the optimality criteria does not require modifying the client application code.

We propose the following IREC-aware segment selection procedure:

(1) The client endpoint looks up for core-path segments to and from the destination ISD where the server AS is located.
(2) For each direction, if there is only one RC, the path service returns the optimal segments computed by the RC. If there is more than one RC, the path service returns *a* segment together with the list of RC identifiers ($\texttt{ID}_{\texttt{RC}}$, cf. Appendix C) existing for each direction of communication.
(3) If the result of the lookup contains a list of $\texttt{ID}_{\texttt{RC}}$s, the client contacts the application server to ask for the desired $\texttt{ID}_{\texttt{RC}}$ out of the provided list. Note that the application developer who controls the server is aware of both the desired criteria for the application and the corresponding RCs.
(4) The client performs another segment lookup for each direction of the communication, providing the path service with the $\texttt{ID}_{\texttt{RC}}$ chosen by the server. The path service returns the optimal segments computed by the specified RC.

E Background on SCION

As this paper designs and builds systems for the SCION Internet architecture, we provide a brief background on SCION.

Architecture. SCION groups ASes in *Isolation Domains* (ISDs). Each isolation domain is organized hierarchically into two levels: (1) Core ASes and (2) Non-core ASes. *Core AS*es, which constitute the *ISD core*, govern the ISD and provide the *non-core* ASes in their customer cones with connectivity to other ISDs. Neighboring ASes can have one of the three following relationships: (1) Core, between two neighboring core ASes, within or across ISDs, (2) Customer-provider, within an ISD, and (3) Peering, between two peer ASes within or across ISDs.

Data Plane. SCION uses packet-carried forwarding state to forward inter-domain traffic. This means that endpoints encode AS-level inter-domain paths into packet headers based on which border router of every AS forwards packets. SCION paths are specified at the granularity of inter-domain (or inter-AS) interfaces representing links between neighboring ASes, providing endpoints with fine-grained control over paths. Border routers do not need to store inter-domain forwarding tables to make forwarding decisions, enabling scalable multi-path forwarding.

Because of the hierarchical organization of ASes in ISDs, SCION paths consist of up to three path segments: (1) an up-path segment, connecting the non-core source AS to a core AS within the same ISD, (2) a core-path segment, connecting the core ASes in the source and destination ISDs, and (3) a down-path segment, connecting the core AS in the destination ISD to the non-core destination AS within the same ISD. A path can consist of just one or two of these path segments depending on the logical location of source and destination ASes in the network. Also, it is possible to use up- and down-path segments partially to reach another non-core AS on the same segment.

SCION paths are reversible: the destination of a packet can reverse the path in the header of the received packet and use the reverse path in the header of the response packet.

Control Plane. Path segments are computed in the *beaconing* process. Core segments are computed by *core beaconing* in which all core ASes participate. Up- and down-segments within an ISD are computed by the *intra-ISD beaconing* in which only the ASes of the same ISD participate. Only core ASes can start a path-segment computation.

To start a core-segment computation, the *beacon service* of a core AS originates a path-construction beacon (PCB) and disseminates it only to the beacon service of its neighboring *core* ASes. The originator encodes an `ASEntry` in the PCB, containing information about its AS hop. This information includes the identifier of the inter-AS interface from which this PCB originated, i.e., the interface at which the path segment arrives at the AS. When the beacon service of a core AS receives PCBs pertaining to another core AS, it selects a subset of such PCBs to be registered at the *path service* of its AS so that the path segment can be looked up and used by endpoints. The beacon service periodically selects a subset of PCBs pertaining to any other core AS to propagate further to its core neighbors. To propagate a PCB, the beacon service appends an `ASEntry` to the PCB that contains information about its AS hop. This information includes the identifier of the inter-AS interface from which the PCB is received and the identifier of the inter-AS interface on which the PCB is propagated. This process computes core-path segments from any core AS to the originating core AS.

Intra-ISD beaconing computes down- and up-path segments in a similar process but with two differences: (1) PCBs travel only from providers to customers within the same ISD, and (2) The beacon service extracts the down segment corresponding to each PCB, reverses them to compute up-path segments, selects a subset of up-path and a subset of down-path segments, registers the up segments

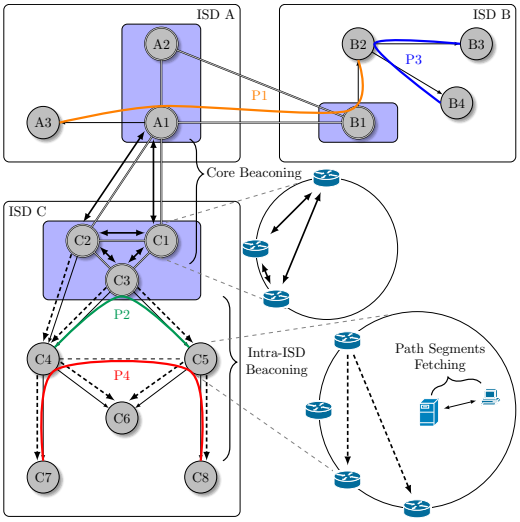


Figure 16: A SCION topology consisting of three ISDs and several possible end-to-end paths.

with its local path service, and registers the down segments with the path service of the originating core AS.

To construct an end-to-end forwarding path, an endpoint needs to request the necessary path segments from the path service at its AS in up to three steps. First, it requests the up-path segments to the core ASes of its ISD. Second, the endpoint requests core-path segments to the core ASes of the destination ISD. When the local path service receives this request from an endpoint, it queries the core path services of the core ASes of the local ISD for core-path segments from each of the core ASes to any core AS in the destination ISD. The local path service sends the responses back to the endpoint and caches them. Third, the endpoint requests its local path service for down-path segments to the destination AS in the destination ISD. When the local path service receives such a request, it requests the core path services of the core ASes in the destination ISD for down-path segments to the destination AS. The local path service sends the responses back to the endpoint and caches them. To reach the core ASes of the destination ISD, the local path service uses the core-path segments it has received in the second step. Once the endpoint has received all the necessary path segments, it combines them into a path and encodes them into data packet headers.

Figure 16 illustrates a SCION topology consisting of three ISDs and several possible end-to-end paths.

F SPC Performance

We evaluate the performance of SPCs by measuring their execution time for the SP described in Section 6.1. Benchmarks are run on standard cloud infrastructure: DigitalOcean CPU-Optimized droplets (32vCPUs, 64GiB RAM) using an Intel Xeon Platinum 8358 @ 2.60 GHz. The ingress and egress gateway are pinned to 16 cores, whereas a single SPC is started and pinned to one specific core.

Our benchmark measures the end-to-end processing delay, including SQLite database accesses and gRPC calls to both ingress and egress gateways. We assume dynamic SPCs that refetch the algorithm on each execution to capture the algorithm retrieval timing. Each SPC processes 1000 SPs, and logs the per-SP latency, reporting the mean and standard deviation per input PCB set size (these are the PCBs the SP will select from). The output PCBs are the selection of 20 PCBs. We group the processing delays into three categories: (1) initialization time, (2) execution time, (3) gRPC and marshaling and (4) the time required to fetch an algorithm.

As shown in Figure 17, the amount of time taken per execution is on the order of milliseconds. Assuming that an AS runs multiple SPCs in parallel, that means thousands of SPs can be executed within a single second. This measurement assumes the worst-case scenario, i.e. there is no algorithm caching, the algorithm is repeatedly just-in-time compiled, and the database used is SQLite and on-disk. An in-memory PostgreSQL database with indices would significantly reduce retrieval latency. As can be seen, the setup overhead for Just-In-Time compilation becomes less significant relative to the execution time as the amount of input PCBs increases. Just-In-Time compiling the eBPF code leads to higher performance from an input PCB size of 32.

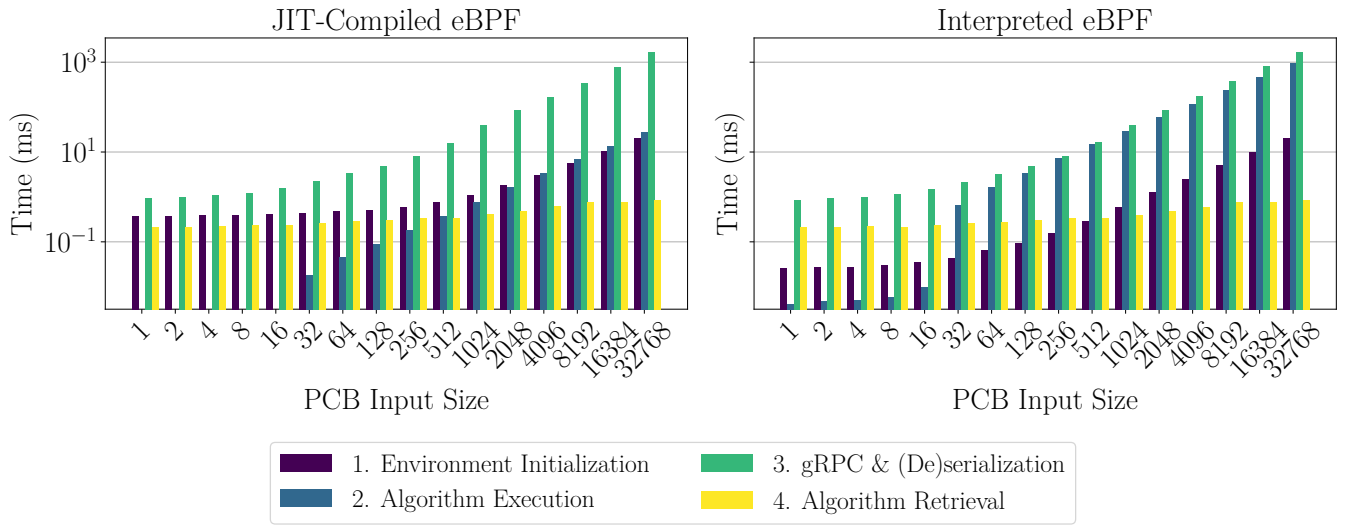


Figure 17: Processing time measured for SP execution in the SPC, comparing eBPF that is Just-In-Time compiled and interpreted.