University of Magdeburg
School of Computer Science

Master Thesis

Deployment of SCION on Internet Exchange Point Infrastructure

Author:
Lars-Christian Schulz

March 18, 2020

Advisors:
Prof. Dr. David Hausheer
Networks and Distributed Systems Lab
Institute for Intelligent Cooperating Systems

Dr.-Ing. Matthias Wichtlhuber
DE-CIX Management GmbH

Für meinen Vater

Abstract

SCION is a novel inter-domain network architecture aiming to be more secure and transparent than today's Internet. It is founded on the principle of making forwarding paths transparent and verifiable to end host, going as far as enabling senders themselves to select a forwarding path. To keep the complexity of path selection under control, paths are constructed from path segments published by autonomous systems (ASes) between the two endpoints. The rules for creating path segments and forwarding paths rely on a topology model reflecting the traditional hierarchy of larger Internet service providers (ISPs) providing service to smaller ones. SCION does model AS peering, but mostly in the traditional sense of interconnecting ISPs of similar size.

ASes of all sizes peer at Internet Exchange Points (IXPs). IXPs flatten the Internet topology by introducing low cost, low latency shortcuts. This work analyzes how well SCION fits an Internet model incorporating IXPs and derives how a SCION deployment in such an environment could look like. In the process, we develop a prototype SCION peering coordinator, taking a role comparable to a route server in BGP.

Acknowledgments

I would like to thank my advisors Prof. Dr. Hausheer and Dr. Wichtlhuber for their guidance in writing this thesis. Their time and advice is much appreciated.

I would also like to thank the SCIONLab team for answering many questions that came up during my work.

Contents

Chapter 1

Introduction

1.1 Motivation

Internet Exchange Points (IXPs) have become an important part of today's Internet infrastructure. They offer customers the opportunity to exchange traffic with hundreds of other networks for a fixed price per port bandwidth. Connecting to a potentially large number of peering partners presents AS operators with the challenge of managing a Border Gateway Protocol (BGP) session for each and every peer. IXP operators such as DE-CIX offer router servers to reduce the number of bilateral BGP sessions and simplify the router configuration for their customers [23].

Participants establish BGP sessions with the route server enabling them to exchange BGP announcements with all other ASes connected to the route server simultaneously. Furthermore, route servers are an ideal place to validate BGP announcements against Internet Routing Registry (IRR) data [14] and perform RPKI [16][18] checks. These checks filter out invalid announcements from misconfigured or malicious border routers, which are possible because BGP itself lacks any authentication and verification mechanisms [3][12].

At the same time, SCION is a new inter-domain network architecture that addresses many of the reliability and security issues of the current Internet [2][21]. Since SCION's control plane messages are cryptographically signed, it is inherently more secure than BGP. SCION's clean-slate approach avoids trying to patch a protocol that was not designed with security in mind. Currently, SCION packets are encapsulated in UDP packets understood by common network hardware. Therefore a link between two SCION ASes over an IXP offering L2 connectivity requirers no special handling on the IXPs side. Nevertheless, a service analogous to a BGP route server might be beneficial to SCION users by providing an interface to establish (peering) links between ASes participating at an IXP.

Route Server in SCION In the current SCIONLab research network [25] ASes are coordinated by the SCIONLab coordinator, a website which provides two interfaces. The first is an "admin" interface to configure "infrastructure" ASes forming the backbone of the network. The second "user" interface allows registered users to create configuration files for "user ASes" which connect to the rest of the network over designated "attachment point" ASes. Whenever a user AS is created or deleted the coordinator updates the configuration of the attachment points [26].

On of the main advantages of using the route server at an IXP is immediate connectivity to a large number of destinations [23]. A "SCION route server" should provide the same advantage, i.e., multi-lateral peering for SCION ASes without the need to configure individual links like in the SCIONLab coordinator. To this end, the "SCION route server" must mediate peering intentions of the participating ASes and interact with the beaconing process SCION uses to establish routes.

Blackholing Another service offered by IXPs is blackholing. Blackholing is a Distributed Denial of Service (DDoS) defence mechanism. To protect against a DDoS attack customers can announce IP prefixes under attack tagged with a special BGP community to the route server, which in turn will make sure traffic to the target subnet is directed to a blackhole destination discarding the traffic and therefore avoiding link and router congestion [6].

SCION was designed with support for DDoS protection. Path announcements in SCION can have a short lifetime (e.g., 10 minutes) limiting the time a path can be used to continue an attack after it was revoked. Since SCION border routers check the validity of routing information in the packet header by verifying the "hop-field" belonging to its AS, traffic using an expired hop-field cannot enter the AS anymore. To avoid congesting the peering link and border router, the IXP's infrastructure could verify the hop-field on behalf of the next-hop router, thereby strengthening the protection from unwanted traffic.

Blackholing has recently been extended to finer grained filtering based on L2-L4 header information to reduce the amount of collateral damage caused by dropping legitimate traffic [7]. This kind of filtering is still compatible with SCION, although on a future native SCION link, the IP header would be replaced with the SCION header posing new challenges for efficient filtering on current hardware. In the future SCION might even eliminate the need for additional filtering by providing source authentication, guaranteed bandwidth allocations and private paths only authorized senders can use.

1.2 Problem Statement and Contribution

IXPs could offer explicit SCION support in the control plane to simplify connecting ASes to the global SCION network and as well in the data plane to fully realize SCION's potential for example in DDoS protection. Both levels of integration come with their own set of questions and challenges.

1.2.1 Control Plane

An IXP SCION coordination service taking the role of the route server requires a new signaling protocol allowing ASes to announce their peering intentions to the coordinator enabling multi-lateral peering sessions. At the same time, the coordination service has to scale to a realistic amount of connected border routers. Introducing multi-lateral peering into SCION also raises questions about the scalability of the beaconing process, since beacons have to be send over every possible path. Therefore, a scalability evaluation is required.

An important aspect of any new signaling protocol is security. It might be possible to leverage SCION's existing infrastructure for signing and verifying control plane messages to this end. In any case, the signaling protocol needs proper AS authentication and verification of announcements to avoid the problems of BGP.

If validation of hop-fields at the IXP is desired, the coordinator also has to be notified of the hop-fields connected border routers accept, so that filtering in the data plane can be configured accordingly.

1.2.2 Data Plane

Freely programmable bare-metal switches offer a readily available high-performance opportunity to work with the SCION packet header. The layout of a SCION packet is illustrated in Figure 1.1. The *forwarding path* is encoded in a series of hop-fields. There is at least one hop-field for each AS the packet passes through. Most relevant to the forwarding of an individual packet is the *current hop-field* which a header field in the *common header* points to. A SCION border router verifies this hop-field by its embedded message authentication code (MAC) and only accepts packets containing a valid hop-field the AS it belongs to has issued.

A bare-metal switch at the IXP could verify hop-fields on behalf of the next border router by looking it up in a list of valid hop-fields in its ternary content addressable memory (TCAM). A challenge with this approach are the limits of currently available bare-metal switch hardware w.r.t. extracting arbitrary data from packet headers. Before data can be match against the TCAM it has to be extracted from the packet. The hardware performing this extraction has limited programmability. Though switches that can extract bits from user defined locations are available, they are still not powerful enough to extract the current hop-field, because its location is determined by the value of another header field that would have to be extracted first. Such a multi-stage extraction is generally not supported for user defined headers. Also note, that user defined header fields are only available on more high-performance switches at all.

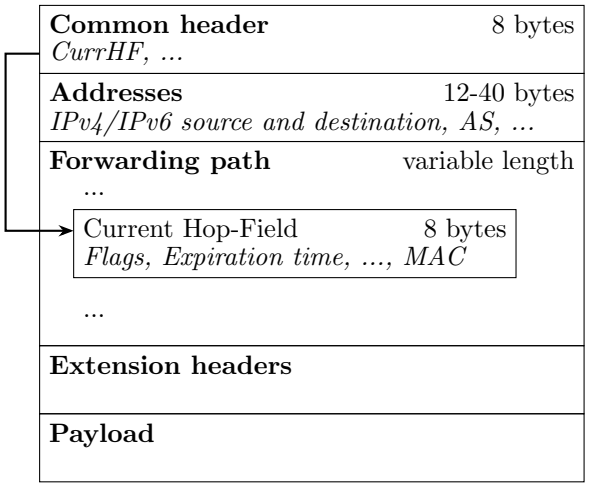


Figure 1.1: Layout of a SCION packet.

As a workaround it would be possible to modify the (software) border routers running in the client ASes to treat IXP peering links specially and copy the necessary hop-field to an agreed upon fixed location. This location could be a standard header field like the IPv6 source address, which should be available for TCAM matching even on low-end bare-metal switches.

Integration of SCION hardware in existing IXP topologies When a bare-metal switch is used at the IXP to process SCION traffic the questions of how to connect to customer SCION ASes and how to integrate said switch with the IXP's backbone arise. One possibility is illustrated in Figure 1.2, where SCION traffic coming from a SCION border router (BR) in AS1 enters the IXP as usual at an edge switch (1). The edge switch separates SCION packets from the normal IP traffic and loops it through a bare metal switch programmed to recognize and deal with the SCION header (2). Valid packets would then continue through the IXP (3) using the usual mechanisms such as MPLS label switching. Additional SCION specific processing can be performed by another (or the same) bare-metal SCION switch when the packet leaves the IXP (4).

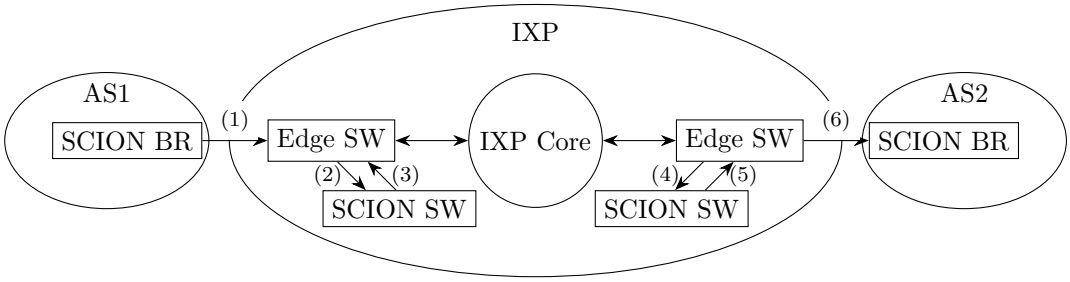


Figure 1.2: Possible topology of a SCION enabled IXP.

Alternatively dedicated SCION ports connecting directly to bare-metal SCION switches at the IXP could be provided. In such an architecture an IXP participant could have a pure SCION-only connection in addition or instead of the traditional link.

1.3 Outline

This thesis focuses on the control plane aspects of deploying SCION at an IXP. The aim is to provide a prototype deployment of SCION at a simulated IXP and characterize its scalability. To this end, the following milestones are defined:

1. *Identification of challenges in deploying SCION at IXPs.* We compare peering in BGP with peering in SCION, especially with respect to open peering at route servers. From this comparison, we derive key challenges in deploying SCION at IXPs and requirements towards a SCION peering coordination service.

2. *Development of a SCION peering coordinator.* We extend the SCIONLab coordinator to support management of peering links between participating user ASes based on simply policy rules.

3. *Simulation of a SCION enabled IXP.* We create a testbed framework enabling us to test the peering-enabled coordinator at a simulated IXP. This framework enables us to evaluate the scalability of the prototype coordinator and the SCION protocol itself in face of a large number of peering connections.

Chapter 2

Background and Related Work

This chapter aims to establish the background necessary to compare SCION with BGP routing and derive requirements for a SCION peering coordination service in Chapter 3.

Section 2.1 establishes some basic routing policies stemming from the hierarchical nature of the Internet and introduces the basic categories peering policies fall in. In Sections 2.2 and 2.3 we review BGP routing and SCION path construction, respectively. Section 2.4 gives an example for routing policies currently available in SCION and Section 2.5 introduces the SCIONLab coordinator.

2.1 Basic Routing Policies

In this section, we review some basic assumptions about typical AS business relationships and the resulting routing policies.

2.1.1 Transit and Peering

The common business relationships between ASes can be broadly categorized into *transit* and *peering* relations. Transit is a service offered by an Internet Service Provider (ISP) allowing their customers to reach the rest of the Internet via the ISP's network. The customer pays its provider per monthly bandwidth, usually measured at the 95^{th} percentile. Transit links form a hierarchy with stub ASes, which have no transit customers, at the bottom and the largest international ISPs, which do not purchase transit from anyone, at the top.

In contrast to transit, peering is often settlement-free, i.e, neither party pays the other. Instead the peering ASes freely exchange traffic deriving about equal value from the arrangement. Since it is essentially "free", peering is preferable over transit. However, it can not fully replace it, because peering in not transitive. Peering ASes can only reach each others networks and the networks of their partner's customers[19].

The hierarchy imposed by the transit and peering model leads to the classification of ASes in three tiers. Tier 1 ASes only sell transit and are fairly restrictive about peering with smaller ASes, since they would prefer them as their customers. Tier 2 ASes both buy and sell transit. Finally, Tier 3 ASes only buy transit and have no customer ASes on their own. Using the topology in Figure 2.1 as an example, `A` and `B` could be Tier 1 ASes, if we assume they are in a peering relation. ASes `C` and `D` could be Tier 2 ASes, also peering with each other, and `E` and `F` Tier 3 ASes. Peering is not restricted to only occur between ASes on the same level of the hierarchy, for example, we could assume that the link between `A` and `D` is also the result of a peering agreement.

Not all AS relationships are covered by the simple model of transit and peering. For example, a provider AS might only advertise a certain subset of all IP prefixes to its customer, only giving access to part of the Internet (partial transit). Another example is a hybrid relation including transit and peering links between the same ASes. Usually a provider AS would not peer with one of its customers, because this enables their customer to avoid using the paid link for some destinations reducing provider revenue. If both ASes are present at multiple colocation facilities, however, it might make sense to have a provider-customer in one and a peer-to-peer relation in another place[11].

2.1.2 Gao-Rexford Model

Gao and Rexford[9] developed guidelines for BGP routing policies guaranteeing BGP convergence. These guidelines were developed to be compatible with the usual transit or peering business models introduced above. The model of AS interconnections resulting from these guidelines, short GR-model, has become a standard for approximating BGP routing. Anwar et al. have experimentally evaluated BGP routing decisions in the Internet and found 64.7% of their observations matching the prediction of the GR-model[1].

The GR-model's guidelines entail two sub-policies concerning the local route preference and to which neighboring ASes routes may be announced:

Local Preference (GR preference) ASes prefer routes towards customers over peering links, since customers pay for the bandwidth, whereas peering links are cost neutral. However, peering links are preferred over routes towards providers, beacuse traffic sent towards a provider incurs a cost to the customer AS.

Route Announcements (GR export) An AS announces routes learned from its customers to its providers and its peers to make the best possible service availabile to the customers. Routes from peers are only exported to customers and not other peers, making peering non-transitive, or providers, which would provide peers with transit service. Routes from providers are only announced to customers, offering them transit, and not to peers, which should not be able to use the AS's provider links.

By following this guideline, the routing becomes "valley-free", i.e., traffic flows up to a provider and then down towards the destination and never down to a customer and then up again.

In a survey conducted by Gill et al. 68% of participating transit providers claimed to use both policies, 19% use GR preference only, and 5% use GR export only, leaving only 6% using neither policy[10]. Therefore, the GR-model appears to be suitable to predict a large portion of BGP routing decisions.

2.1.3 Basic Peering Strategies

PeeringDB[20] is a user-maintained database where participating ASes present their type of business (enterprise, content provider, etc.), traffic volume and peering policy among other information helpful for selecting peering partners. PeeringDB is widely used by network operators and has been found to reasonably accurately represent network properties[17].

Individual peering policies vary between ASes, but PeeringDB categorizes them into three general strategies: Open, selective, and restrictive. ASes following an open policy are generally open to peer with any interested party at any location both ASes are present in. A selective policy indicates, that there are some requirements to fulfill, like a maximum or minimum in/out traffic ratio. Nevertheless selective ASes are open to peering with everyone meeting the prerequisites. A restrictive policy means, that the AS is generally not inclined towards peering. Operators with a restrictive policy will peer only with very selected partners and do not participate in open peering discussions. We will use this general classification as a basis for deriving requirements for a SCION peering coordinator in Section 3.3.1.

2.2 BGP

The Border Gateway Protocol (BGP) is the Internet's de facto exterior gateway protocol. It is universally employed to exchange reachability information between interconnected autonomous systems (ASes). As such it is the main control plane protocol spoken at IXPs. Understanding how BGP is used by IXP participants will be an important basis for discussing how SCION might complement or replace it in the future.

2.2.1 BGP Basics

BGP is defined in a number of RFCs. It has gone through multiple revisions and extensions. The current version is BGP-4[22], which we will refer to simply as BGP. In the following we briefly discuss some basic aspects of BGP to prepare the comparison with SCION in Section 2.3.

Neighboring ASes maintain *BGP sessions* with each other. A BGP session is established over a TCP connection. When establishing a new session two BGP routers exchange the routes they each selected for advertisement to other routers. After the initial exchange updates are only sent if routing information changes.

The pivotal type of message sent by BGP speakers is the `UPDATE` message. An `UPDATE` message contains a (possibly empty) set of advertised routes and a (possibly empty) set of withdrawn routes. A route is identified by the IP address prefix of the reachable destinations. An `UPDATE` message advertising new routes contains a set of path attributes common to all prefixes. Among the mandatory path attributes is a sequence of AS path segments which traces back the chain of ASes trough which the advertisement has passed. ASes are identified by their AS number (ASN). AS numbers are assigned by regional Internet registries (RIRs). Another mandatory path attribute is the IP address of the router that should be used as next hop towards the destination should the path be selected for forwarding packets.

A BGP speaking router collects information received from its peers in a Routing Information Base (RIB). Based on the data in the RIB the router selects paths and the reachability information to propagate to its peers in turn. The border router is then free to build its routing table from the paths learned over BGP moderated by its local policies.

BGP Path Construction Since IP prefixes originate from the AS owning the corresponding address range and are propagated to more and more distant ASes, BGP paths construction starts at leaf ASes. Consider the example in Figure 2.1 where AS `E` announces a prefix to its transit provider `C`. `C` in turn announces reachability of `E` to its other neighbors `A`, `D`, and `F`. `D` receives routes from multiple other ASes: From `B` over `A` and `C` and from ASes `A` and `C` directly. `D` decides, according to its policies, which route to advertise to its client `F`. In this example, `D` decides to offer the connection over its link to `C`. This would be a typical decision if the link between `C` and `D` is a peering link which does not incur a traffic dependent cost to `D`. Additionally `F` receives an update from `C` directly. `F` can then select which of the two links to use for reaching `E` or split its outgoing traffic between the links.

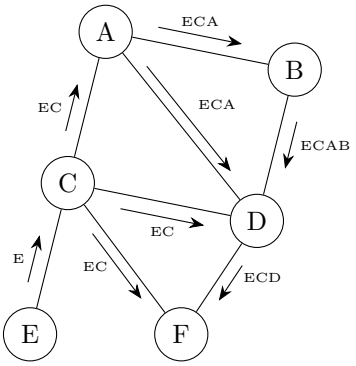


Figure 2.1: BGP route update originating from AS E.

Since BGP routes depend just on the IP destination, only destination based policies are expressible. We will compare the set of policies BGP allows with the policies possible in SCION in Section 3.1.

BGP Communities BGP prefix announcements can be tagged with a set of BGP communities. Communities allow to mark a set of routes with some common attributes like policy information. For example, if a route is only available to educational networks, it could be tagged with a BGP community signaling other routers to only consider the route for connecting educational networks.

There are a few "well-known" communities any community-aware BGP router knows, but most communities are special values cooperating network operators agreed upon. Among the well-known communities are `NO_ADVERTISE` (encoded as 0xFFFFFF02) and `NO_EXPORT` (encoded as 0xFFFFFF01). `NO_EXPORT` and `NO_ADVERTISE` forbid advertisement to BGP neighbors in other ASes and to any BGP neighbors at all, respectively. Communities originally were 4 byte values[4], but were later extended by extended communities[24] and large communities[13].

The route servers provided by IXPs generally allow some control over prefix distribution by the originating AS via BGP communities. Some example communities understood by the route server operated by DE-CIX in Frankfurt are listed in Table 2.2.

Action	Community
NO_ADVERTISE (well-known)	65535:65281
NO_EXPORT (well-known)	65535:65282
Do not redistribute prefix to AS `PEER-AS`	0:`PEER-AS`
Redistribute prefix to AS `PEER-AS`	6695:`PEER-AS`
Do not redistribute prefix to country `COUNTRY`	65223:`COUNTRY`
Redistribute prefix to country `COUNTRY`	65213:`COUNTRY`
Add NO_ADVERTISE when redistributing	0:6695
Add NO_EXPORT when redistributing	6695:6695

Figure 2.2: Some operational BGP communities available at DE-CIX route servers[5].

2.2.2 Route Server

The primary reason for joining an Internet Exchange Point is to gain as many new suitable peering connections as possible. Therefore the value of an IXP increases with the number of participants it already has, creating a positive feedback effect. Large IXPs interconnect hundreds of ASes, meaning that getting the full value of an IXP connection requires a new participant to manually set up hundreds of BGP sessions creating large administrative overhead.

Each BGP sessions constitutes a bilateral peering link. While bilateral peering might be suitable for large well established ASes, IXPs attract all sizes of AS operators. Therefore, many IXPs offer an alternative to bilateral peering in form of multilateral interconnection coordinated by a route server. A route server is a BGP brokering system, that only participates in the control plane, but not in the data plane. It maintains a single BGP session with every AS willing to engage in multilateral peering. These route server clients announce routes to the route server, which then selects and re-announces suitable routes to its clients. In contrast to a regular BGP router, the route server does not insert itself as next hop, and strips the IXP's ASN from the path.

In this manner, all clients connected to the route server can exchange traffic after configuring only a single BGP session.

Besides the scalability problem route servers can solve another problem, they can filter announcements from peers to avoid propagation of invalid prefixes. Filtering typically involves removing too long or too short prefixes, as well as private and reserved prefixes (bogons/martians), and validating route origins against RPKI (Resource Public Key Infrastructure). RPKI provides a verifiable database mapping IP prefixes to ASes which are allowed to originate routes to that prefix[18].

Looking Glasses BGP looking glasses are tools for debugging of route announcements. Looking glasses provide a user interface for executing queries against a router's RIB. Looking glasses are available at some IXPs providing insight into their route servers. Available information includes which routes peers advertise, which of these routes were accepted or rejected, and to whom accepted routes are exported.

2.2.3 Blackholing

Blackholing is a DDoS mitigation service available at some IXPs. Blackholing can not only protect end hosts under attack, but also network infrastructure and the link to the IXP itself by filtering traffic within the IXP. It is activated by announcing the IP prefix under attack with a special BGP community. Border routers participating in the blackholing system will then route traffic destined for the blackholed prefix to a special next hop IP, that is assigned to a special blackhole MAC address. The IXP drops all traffic addressed to the blackhole MAC on ingress, preventing the link to the AS under attack from becoming congested [6].

Blackholes can also be announced over route servers. By tagging them with a set of BGP communities, the route server can be instructed to distribute the blackhole announcement to a specific set of ASes, as to not disturb the multilateral peering with ASes sending legitimate traffic.

2.3 SCION

SCION is an inter-domain network architecture aiming to improve the Internet's scalability, reliability and security. SCION groups AS into *isolation domains* (ISDs) which share a common policy, the *trust root configuration* (TRC). The TRC serves as root for the trust infrastructure validating cryptographic keys and addresses assigned to member ASes. A subset of the ASes in an ISD form the *ISD core*. ASes in the core are called *core ASes*. The core ASes are responsible for managing and distributing the ISD's TRC. They also provide connectivity to the core ASes of other ISDs.

A SCION AS is uniquely identified by a 6 byte AS number. Often, the AS number is combined with a 2 byte ISD number, together forming an 8 byte identifier. The textual representation of an ISD-AS number pair separates the ISD and AS number with a hyphen. The ISD number is written in base 10 and the AS number in base 16, separating 2 byte groups with colons similarly to how IPv6 addresses are formatted[15]. For example, `10-ff00:0:1` is an AS in ISD 10.

An AS can be part of multiple ISDs by using a different identity (i.e., AS number) in each one. This way it can even serve as core AS in some ISDs and as a non-core AS in others.

Link Types SCION not only distinguished different types of ASes, but also different types of links. The link type plays an important role in SCION's path exploration process referred to as *beaconing*.

There are three link types: Core links, provider (parent-child) links, and peering links. All link types connect exactly two ASes with each other. Core links are symmetric links between two core ASes. They can connect core ASes of the same ISD or of different ISDs. Provider or parent-child links connect ASes of the same ISD. They are asymmetric and typically drawn as directed edges in diagrams. The parent-child links of an ISD must not contain cycles. Peering links are symmetric like core links, but connect two non-core ASes in the same or different ISDs.

Parent-child links model a classical provider-customer relationship between two ASes. The customer buys connectivity from the provider to gain membership in an ISD and connect to the wider network. Peering links model a peering relation in which both ASes gain from exchanging traffic. Core links connect core ASes, which are expected to be the largest carriers of their respective ISDs. As such, they only sell connectivity within their ISDs and would often have peering arrangements with other core ASes.

2.3.1 Beaconing

In contrast to BGP, the path exploration process of SCION starts at the core ASes. The *beacon servers* of core ASes generate *path construction beacons* (PCBs) in regular intervals and send them to all neighboring core ASes. The PCB records the chain of ASes it has passed through. When a core AS receives a beacon from another core AS, it adds itself to the beacon and disseminates it to its other neighbors. Using these PCBs, the core ASes learn paths between them. Path segments between core ASes are called *core path segments*.

The intra-ISD beaconing process between core and non-core or two non-core ASes is different from the inter-ISD beaconing process between core ASes described above. Intra-ISD beaconing follows provider links and does not cross ISD boundaries. At regular intervals, core ASes send

beacons to their children, which then add themselves to the beacons and send them to their children in turn. There must be no loops during inter-ISD beaconing, i.e., the parent-child links must form a directed acyclic graph. Via intra-ISD beaconing, the ASes learn paths from the core ASes to the leaf ASes from which *up* and *down path segments* are generated.

Both types of beaconing use the same PCB format. In addition to the ASes the beacon passes through, it also records the interface at which it arrives and leaves an AS. Furthermore, it has to record information on the peering links an AS provides, because beacons are not send over peering links. This information is used during end-to-end path construction (see Section 2.3.3).

Example Figure 2.3 shows the same topology as Figure 2.1 implemented in SCION. All ASes are assumed to belong to the same ISD. The Tier 1 ASes `A` and `B` have become core ASes. They provide service to `C` and `D`. Note that the link between `A` and `D` is a provider-customer or parent-child link instead of a peering link. In SCION a regular AS cannot peer with a core AS, hence a SCION peering link is not possible. Of course, the business relation between `A` and `D` might still be that of peering partners, but the SCION link cannot reflect this without additional policies.

Beacons are sent from `A` to `C`, which sends beacons to `E` and `F`, and from `A` to `D`, which sends beacons to `F`. Additionally, `B` sends beacons to its only child `D` which continue on to `F`. That means, `F` is receiving beacons for the path segments `A-C-F`, `A-D-F`, and `B-D-F`. `F` can make all these paths available to its end hosts. No beacons are transmitted over the peering link between `C` and `F`, instead both `C` and `D` record the link's presence in the beacons they send to their children.

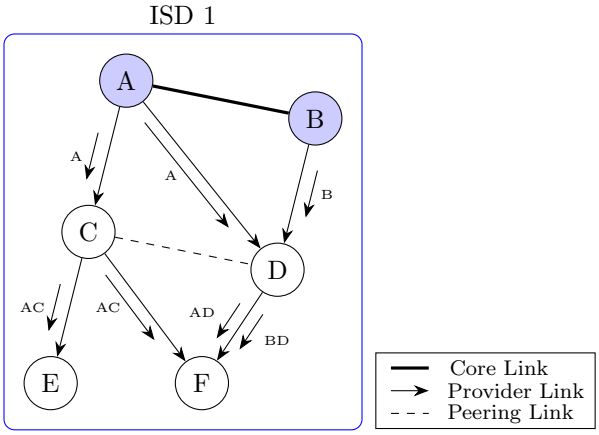


Figure 2.3: Beaconing in SCION.

2.3.2 Path Segment Registration

The core, up-, and down-segments created by beaconing are registered at *path servers* from where they can be retrieved by end hosts to build a path to a destination AS. Every AS has at least one path server. Core- and down-segments are registered at core path servers, up-segments at the local path server. Each AS decides which paths it wishes to register as path segments based on local policies. Up-segments allow an end hosts within the AS to reach a core AS. Down-segments allow end hosts in other ASes to reach the AS from one of the core ASes in the destination ISD. If an up- and down-segment do not share the same core AS, a core-segment is used to link the two (given a suitable core segment has been registered at the core path servers).

Selecting which path-segments to register at a path server is a natural point to apply filtering rules enforcing AS policies. For example, by not registering a path segment as a down-segment, an AS operator can ensure, that no traffic destined for their AS will arrive on this path. Traffic for child ASes might still use the path though, because child ASes learn of its parents links from the beacons it sends them. If this is not desired, the solution, in this case, is to filter the outgoing beacons. We discuss the current state of path policies in SCION and elaborate further on this example in Section 2.4.

Paths segments not registered at the path servers can still be used for constructing forwarding paths. The host initiating the connection simply has to obtain the segments by some other means. Paths not available from the public path servers are called *hidden paths*. They allow AS operators to reserve paths for special clients to increase availability and confidentiality of their connections.

2.3.3 Path Construction

In SCION, the forwarding path for a packet is selected by the end host sending it. Every packet contains the entire forwarding path. The set of paths an end host can select from is determined by the segments the ASes involved chose to register. In the following, we discuss how end hosts form forwarding paths from path segments.

The first step is retrieving path segments from a local path server. Up-segments are supplied by the local path server directly, core- and down-segments toward the destination are retrieved from core path servers and cached on the local path server.

A forwarding path can at most consist of one up-, one core, and one down-segment. For example, consider the topology in Figure 2.3. The path `E-C-A-B-D-F` from `E` to `F` requires an up-segment `E-C-A` to reach the ISD core, a core segment `A-B` and a down-segment `B-D-F` to arrive at the destination. While a path can consists of all three segment types, it can contain less. For example, a shorter path connecting `E` and `F` would be `E-C-A-D-F`, omitting a core path segment.

There is always a path traversing a core AS between any two non-core ASes, but the path can be cut short if the up-, and down-segment share a peering link or overlap. An example for the first case would be the path `E-C-D-F` utilizing the peering link between `C` and `D`. The second case is illustrated by the path `E-C-F` constructed from the up-segment `E-C-A` and the down-segment `A-C-E` both passing `C`.

A path may only contain a single segment if the source and destination AS lie on the same path segment, e.g., there is a valid path from `C` to `E` directly or from `A` over `C` to `E`. Another case of a single path segment is when both end hosts lie in core ASes.

The topology in Figure 2.3 contains only a single ISD, but the process is very similar when multiple ISDs are involved. In that case, the core path segment spans multiple ISDs and a peering link between two on-path ASes in the two different ISDs might be used to create a shortcut.

An important limitation of SCION paths is that only a single shortcut over a common AS or peering link can be used. If the up- and down-segment cannot be connected in zero (common AS on both) or one (peering link) hop, the path has to include a core AS. We analyze the implications this has on a deployment of SCION at an IXP in Section 3.2. More details on path construction, as well as the precise algorithm are given in the SCION book Section 8.3[21].

2.4 SCION Path Policies

In SCION, some basic routing policies can be expressed by simply selecting the appropriate link type, i.e., defining a link as either customer-provider or peer-to-peer, with similar effects to the GR export policy. More complex policies require a more refined approach. The SCION book[21] introduces the following approaches in Section 10.9.2:

Beaconing Control Beacons destined for child ASes can be filtered to only convey desiered upstream paths and only have to include the peering links an ASes whishes specific customers to see.

Path Policy Beacon Extension Explicit policy definitions, like a list of downstream ASes which are or are not allowed to use the path segments constructed from the beacon, are added as a beacon extension.

Hop Field Encryption Hop fields can be encrypted, making the corresponding path unusable until the path is activated by a special packet containing the entire end-to-end path, which can be inspected and subjected to policies by all ASes along the path.

To our knowledge, only the beaconing control approach has been (partially) implemented yet. The beacon server supports a policy file controlling beacon propagation to child ASes, which allows

filtering for the maximum path length and ISD loops on the path, as well as blacklists of ASes and ISDs which must not occur in the beacon. An example policy file filtering all beacons containing AS `ff00:0:110` or ISD 2 is shown in Listing 2.1. At the moment, only a single policy is supported, which is shared by all child ASes. Also, there is no control over the selection of peering links to be included in the beacons.

Listing 2.1: Example for a beacon propagation policy file.

```
1 BestSetSize: 5
2 CandidateSetSize: 100
3 MaxExpTime: 63
4 Filter:
5   MaxHopsLength: 10
6   AsBlackList: ["ff00:0:110"]
7   IsdBlackList: [2]
8   AllowIsdLoop: false
9 Type: "Propagation"
```

In addition to the beacon propagation policy, the path segment registration can be configured with similar policy files. Figure 2.4 shows an example topology we will use to explain the possibilities of the beacon server policies. We provide the beacon server of AS `1-ff00:0:130` with a beaconing, an up-segment registration, and a down-segment registration policy. The beaconing policy filters out AS `B`, the up-segment policy filters out AS `C`, and the down-segment policy filters out AS `D`.

Because of the beacon propagation policy, `F` only receives beacons containing paths over `C` and `D`. Therefore, `F` can reach `A` over the paths `F-E-C-A` and `F-E-D-A`, but not `F-E-B-A`. The same paths are available in the reverse direction.

Disallowing registration of paths containing `C` as up-segments, means that only the paths `E-B-A` and `E-D-A` exist to reach the ISD core from AS `E`. Note that this policy has no bearing on the paths available the children of `E`, who still can construct paths over `C`.

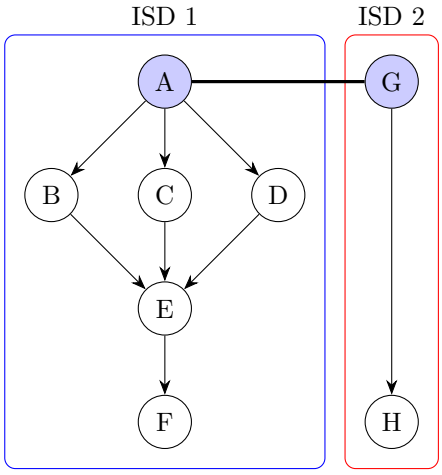


Figure 2.4: Example topology for beacon server policies. The beacon server of AS `E` is configured to filter out beacons containing AS `B`, up-segments containing AS `C` and down-segments containing AS `D`.

Since different filters were applied to up-, and down-segment registration, path construction becomes asymmetric. Consider the possible paths between `E` and `H`. An end host in `E` can construct the paths `E-B-A-G-H` and `E-D-A-G-H` to `H`. An end host in `H` can also use `B` has a hop yielding the path `H-G-A-B-E`, but not `D`, since `D` was filtered out of down-segment registration. Instead, a down segment over `C` is available (`C` is not used in up-segments, but in down-segments), yielding the path `H-G-A-C-E`. Note that even though different paths segment are available depending on which host initiates a packet flow, the destination host can still respond to the source on the same path by

reversing the order of hop-fields in the SCION header.

In addition to up-, and down-segment policies, there are core-segment policies only relevant for core beacon servers, as well as hidden path policies. The full configuration for this example is listed in Appendix A.3.

2.5 SCIONLab Coordinator

The SCIONLab coordinator[26] is a web application built with the Django application framework[8]. It manages the SCIONLab research network by generating configuration files for all participating ASes. ASes can automatically fetch their configuration through a RESTful API.

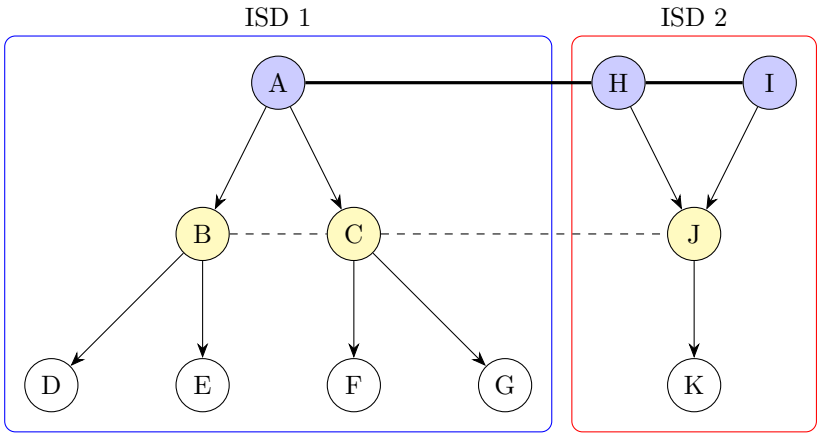


Figure 2.5: Possible topology managed by the SCIONLab coordinator with two ISDs. `A`, `H`, and `I` are core ASes. `B`, `C`, and `J` are attachment points. The user ASes `D`–`G` and `K` are each connected to attachment points via parent-child links. Additionally, the attachment points have peering connections.

The coordinator distinguishes two types of ASes: Infrastructure and user ASes. Infrastructure ASes form the backbone of the network and provide connectivity to user ASes. They are configured in the coordinator's administrator interface, which uses Django's capability to automatically generate an admin web-interface based on database models. All core ASes are infrastructure ASes, but non-core ASes can also belong to the infrastructure. Some infrastructure ASes are designated as *attachment points* at which user ASes attach to the network. Every user AS attaches at exactly one attachment point.

User ASes can be created by users who registered through a public interface. Every active user AS connects via exactly one attachment point. User ASes created through the user interface cannot have child ASes or peering links. Nevertheless, such a configuration is possible by manually editing the user AS in the administrator interface. Infrastructure ASes must have a static IP routeable in the Internet. User ASes can alternatively connect via a VPN tunnel to their attachment point.

Figure 2.5 shows an example of how a topology managed by the coordinator can look like.

Chapter 3

IXP Peering in SCION

In the first section of this chapter, we compare how peering is integrated in SCION with how typical peering sessions are configured in BGP. We identify peering policies which are easily expressible in SCION and policies that cause issues. Section 3.2 examines one of the more difficult cases, transitive peering, in greater detail.

In Section 3.3, we present a possible extension of the SCIONLab coordinator to manage peering between user ASes based on simple routing policies. The implementation of these extensions is the topic of Section 4.2 in the next chapter.

Finally, Section 3.4 presents reasons to integrate SCION in an IXP's data plane and how such an integration can be achieved with bare-metal switches.

3.1 Comparison of SCION and BGP Peering

With the background established in Chapter 2 in mind, we can compare SCION and BGP in terms of peering configuration and possibilities. We will use the SCION topology in Figure 3.1 as a running example throughout this section. The files necessary to run this topology are provided in Appendix A.1. The example topology will illustrate forwarding paths that are possible in SCION now, that might be possible in future implementations, and that are not allowed in the SCION architecture at all.

The example topology contains two ISDs. AS `A` serves as core AS in ISD 1. The core ASes of ISD 2 are `H` and `I`. The core ASes are linked by (symmetric) core links. Recall that the core links carry inter-ISD beacons. Core ASes provide transit service to non-core ASes in their ISDs. In ISD 1, `A` provides transit to `B` and `C`. In ISD 2, `H` provides transit only to `J` and `I` to `J` and `K`. The remaining ASes are leaf ASes not selling transit to other ASes, instead just buying connectivity from others. Provider-customer relationships are modeled as directional links. Functionally, these links carry the intra-ISD beacons from which up- and down-path segments are constructed (see Section 2.3.1).

The example contains six SCION peering links (dashed lines). Within ISD 1, `B` and `C` are peering, as well as `D` with `E`, which in turn peers with `F`. Peering links can cross ISD boundaries. In the example, this happens in three cases, between `C` and `J`, between `G` and `K`, and between `D` and `J`. The peering link between `D` and `J` also illustrates how peering links provide the highest flexibility in breaking the hierarchy of provider-customer links by interconnecting ASes in different ISDs and at different levels of the hierarchy. Remember though, that peering links do not carry beacons and are not a substitute for parent-child links, that do.

In the following, we will discuss how the equivalent of Figure 3.1 can be created with BGP and how routing policies can be applied in BGP and in SCION.

3.1.1 GR Export

Ignoring multi-path routing, the BGP equivalent of Figure 3.1 can be created by configuring all routers to follow the GR export policy. SCION parent-child links must be treated as connecting a transit provider to one of its customers. Thus, BGP routers following the GR export policy should

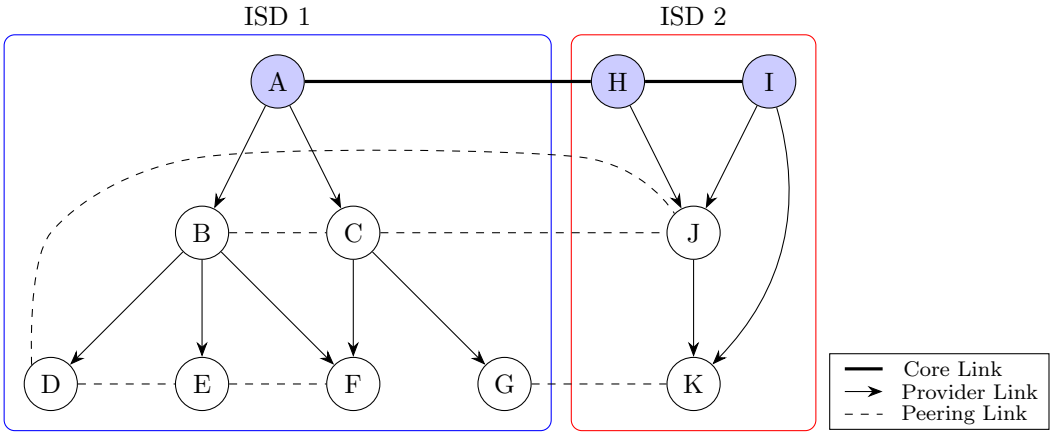


Figure 3.1: Example topology with two ISDs.

announce prefixes learned from their parent ASes to their child ASes, but not to other parents. Prefixes learned from child ASes must be announced to the parent ASes and other children. Taking AS `C` as an example, routes from `C`'s provider `A` are announced to `C`'s customers `F` and `G`. Customer routes from `F` are announced to `A` and to the other customer `G`. The same applies to `G`.

SCION peering links must be treated the same as peering links in BGP. Prefixes learned from peers are only announced to children, not to parents or other peers. Prefixes learned from children are announced to peer ASes. Again using AS `C` as example, routes learned from `B` (to `D`, `E`, and `F`) and routes learned from `J` (to `K`) are announced to `F` and `G`. Routes to `F` and `G` are announced to the peers `B` and `J`.

Announcing a route to a customer AS is similar to sending a beacon containing the route. The beacon allows downstream ASes to construct up-segments from the beacon to reach the ISD core and consequentially the whole Internet. Announcing a route to a provider AS is similar to registering a down-path segment at a SCION core path server. It enables remote ASes to select a path to the announced destination passing through the announcer. The difference is, that the decision to register a down-path segment is made by the destination AS, not by one of its upstream providers.

Figure 3.2 shows two forwarding paths possible with this configuration and an impossible path. The red path from `D` over `B` and `C` to `G` is possible, as is the green path `D-B-A-H-J`. Both of these path have the property, that packets first flow up (up-segment, in reverse direction of parent-child links) and then down (down-segment, in parent-child link direction), possibly with a core-segment or a peering link in the middle. Paths like the blue one (`C-G-K-J`), where packets flow down and then up again, are not possible. Not in BGP with the GR-export policy, since `C` did not receive a reachability announcement for `J` from `G`, and not in SCION, because there are no suitable path segments.

This property is called "valley-free" routing and occurs naturally in SCION, in BGP it guarantees route convergence. It generally conforms with ISP business models. An ISP only provides transit service to paying customers, not to peers or its own providers. Forwarding traffic between two providers would mean paying both of them for providing a service to them. Therefore SCION's inability to provide such paths does not appear as a limitation.

3.1.2 GR Preference

A big difference that remains between the SCION topology in Figure 3.2 and the BGP equivalent outlined in the previous section, is that SCION is a multipath protocol. Two SCION hosts in different ASes can use multiple forwarding paths in parallel to increase availability and bandwidth. Additionally, the final selection of which forwarding path to use is made by the end hosts, not by routers. If, for example, an end hosts in AS `G` whishes to communicate with a destination in AS `K`, it can use the shortest path (in hops) over the peering link between `G` and `K`. However, because the path is constructed from the up-path segment `G-C-A`, core segment `A-H`, and down-segment `H-J-K`,

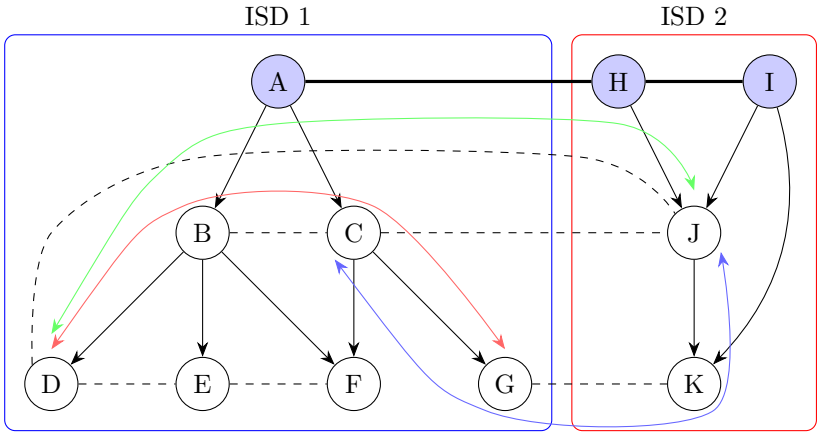


Figure 3.2: Example forwarding paths. The red and green path are valid in SCION, the blue one is not.

the long route `G-C-A-H-J-K` is another valid selection. Because `C` and `J` also share a peering link, there is a third valid path using the same path segments: `G-C-J-K`. A fourth and fifth valid path are `G-C-A-H-I-J-K` and `G-C-A-H-I-K`. The source host in `G` is free to select any single or multiple of these options to reach `K`.

Multipath communication and end host based path selection are at odds with the GR preference policy. According to the GR preference policy, links to children (customers) should be preferred over peering links, which should be preferred over links to parents (providers). This policy directly reflects, that sending traffic to a customer makes an AS money, sending traffic to a peer is usually cost neutral and sending traffic to a provider costs money. In our example, `G` would probably prefer the end host using the direct path `G-K` only, because any other path involves directing traffic trough `G`'s provider `C`. In BGP, the preference for cheaper paths is realized by assigning a higher Local Preference to cheaper routes. In SCION, `G` cannot prevent the paths `G-C-J-K` and `G-C-A-H-J-K` from being used, because the up- and down-path segments between `A` and `G` must be registered to connect to the rest of the Internet.

`G` can try to steer connections to use the peering link by providing more bandwidth along the peering link to `G` than to its provider `C`. However, `G` has to provide a lot of bandwidth on the link to `C`, because every AS except `K` is only reachable over this link. Therefore, the longer paths might stay an attractive option for end hosts in `G` and `K`.

The SCION book[21] points out the issue outlined above and offers a few suggestions on how to circumvent them (Section 10.9.5):

- Bandwidth could be allocated per flow by inspecting source and destination AS. For example, `G` could limit the bandwidth available through `C` for packets destined for `K`.
- Hop-field encryption can be used to prohibit unwanted paths.
- SCION connections might simply cost more than traditional IP service to compensate for the less lucrative paths.
- Pricing could be calculated per-path.

Harmonizing multi path communication with the business model of peering ASes will surely be the subject of future research, but is out of scope for this thesis.

3.1.3 Selective Route Availability

An AS might want to make certain transit or peering links available only to selected customers, keep them for end hosts in its own network or only use them as backup links in case another connection fails. In BGP, this is possible by selecting the routes to announce to customer ASes and in the internal network appropriately.

As discussed in Section 2.4, SCION ASes can keep links hidden from child ASes by not including the links in beacons sent to that AS. An even more powerful mechanism are hidden paths. Recall from Section 2.3.2, that path segments can be transmitted out of band to selected ASes or hosts without registering them at a path server. Only hosts who possess the hidden path segments can use the associated routes, for example as a backup path. There is no equivalent to hidden paths in BGP.

3.1.4 Partial Transit

In BGP, it is possible to announce a selected subset of prefixes to a customer, instead of a default route to arbitrary destinations, thus only providing partial transit to the customer. For example, a multi-homed AS like `F` might only receive full transit from `C`, while `F`'s second provider `B` only provides routes within ISD 1, but for a cheaper price than `C`.

Basic SCION has no equivalent to partial transit in BGP. It would be possible to restrict for which destinations a path segment can be used by including additional metadata in an extension to SCION's path construction beacons, but such an extension is not available yet.

3.1.5 Core AS Peering

SCION's core AS links exist solely between core ASes. At the same time, core ASes cannot peer with non-core ASes. Normal peering links can only be created between non-core ASes. Core links model peering between Tier 1 ASes in today's internet. Traditionally Tier 1 ASes would only peer with other Tier 1 ISPs, justifying the omission of core to non-core peering to some extent.

In case peering between a core and a non-core AS is required, there are two possibilities: The core AS could instead have a parent-child link to the non-core ASes, or the core AS exists a second time in the topology as a non-core AS with a different AS number. The former option only works, if both ASes are in the same ISD and requires additional path policies to emulate the behavior of a peering link. The later option is always available, but introduces additional complexity for managing multiple AS identities.

Example Assume AS `D` of our example topology wants to peer with `A`. Because `A` is a core AS and `D` is not, a peering link is not possible. Instead, `A` can become a parent of `D` (Figure 3.3). The difference to a peering link is, that with no additional policies, `D` can now not only reach `A`'s other children over the new links, but also `H` and the rest of ISD 2 as if the peering link were transitive.

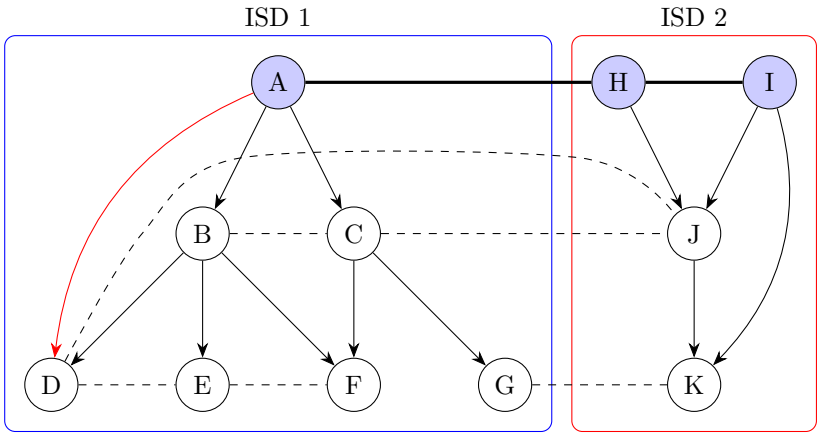


Figure 3.3: Non-core AS `D` peering with core AS `A` in the same ISD.

If `D` wanted to peer with `H`, the workaround of swapping a peering link with a parent-child link does not work anymore, because `D` and `H` are in different ISDs. Instead, a new AS `H'` is created as a non-core AS sharing the same internal network as `H` (Figure 3.4). `H'` must have the same child ASes as AS `H` to provide all paths possible when peering with `H` itself.

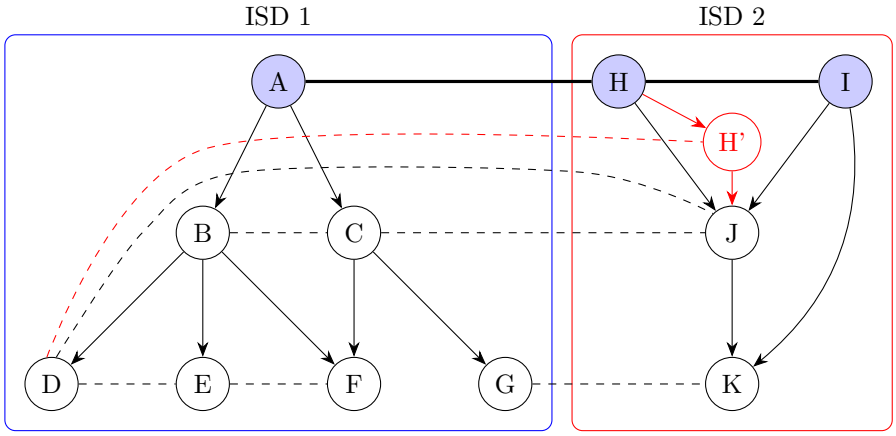


Figure 3.4: ASD peering with core AS `H` represented under the second identity `H'`.

3.1.6 Peering with Direct or Indirect Customers

AS operators tend to avoid peering with direct or indirect customers, because the free peering link would allow traffic to bypass the paid provider-customer link. Nevertheless, hybrid relationships where one AS offers another both transit and peering at different points of presence have been observed[11].

Peering links between customer and direct or indirect providers are possible in SCION. We will illustrate the paths such peering links enable with the topology in Figure 3.5. AS `B` is in a hybrid relationship with AS `E`. They share a provider-customer and a peering link. Additionally `B` peers with its indirect customer `F`. `F` also peers with `C`, which is its indirect provider over `E`. With this setup, `F` can communicate with `D` over the paths `F-E-B-D`, using the peering link between `E` and `B`, and `F-B-D`, using the peering link between `F` and `B`. `F` cannot, however, reach `A` on a path involving a peering link. The only valid paths between `A` and `F` are `A-B-E-F` and `A-C-E-F`, because SCION can only build paths with peering shortcuts between up- and down-segments. Peering links cannot be used as shortcut within a segment. This way, `F` cannot use its peering links to `B` and `C` as a substitute for the parent-child links.

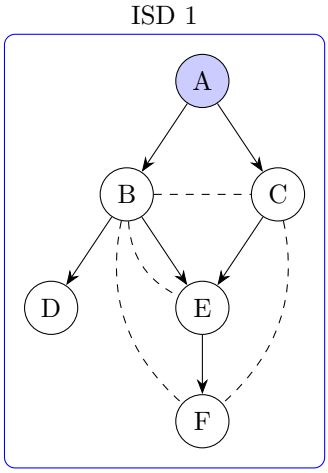


Figure 3.5: `B` and `C` are peering with direct or indirect customer ASes.

The definition of the example topology in Figure 3.5 is available in Appendix A.2.

3.1.7 Transitive Peering

Peering is usually not transitive. Following the GR export policy, routes learned from peering partners are not announced to other peers. Peering links in SCION exhibit the same behavior. A SCION forwarding path can contain at most one peering link, therefore it is fundamentally impossible to use a peering link transitively.

Figure 3.6 shows two peering paths that are not valid in SCION. For the red path, AS `C` would allow its peers `B` and `C` to exchange traffic over its network. AS `E` does the same with `D` and `F` to enable the blue path. We will discuss possible workarounds for this limitation in Section 3.2.

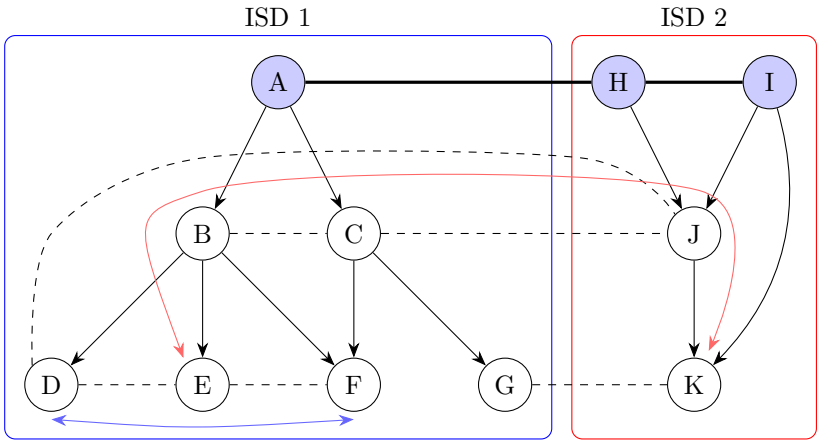


Figure 3.6: Transitive peering example paths.

3.2 Transitive Peering

Peering links in SCION are not transitive, i.e., AS `A` peering with `B`, which peers with `C`, does not mean that `A` can also exchange peering traffic with `C`. This matches the general definition of peering in today's Internet. However, with BGP it is possible for `B` to allow traffic exchange between `A` and `C` over its network. In this section, we are sketching a few ideas of how transitive peering could be achieved in SCION.

Figure 3.7 shows part of a topology with the four user ASes `D`, `E`, `F`, and `G`. There are peering links between `D` and `E`, `E` and `F`, and `F` and `G`. The red forwarding path between `E` and `G` is not available in SCION, because a forwarding path must not contain more than one peering shortcut. Assume ASes `E` and `F` allow each other, as well as their neighbors `D` and `G` to reach the other peers over their peering links. How can we modify to topology to enable communication between all user ASes without involving their parents `B` and `C`?

If all user ASes participate at the same IXP, the most obvious solution is to add the missing pairwise peering links `D` to `F`, `E` to `G`, and `D` to `G` (draw as dotted lines). This way, we have a fully meshed network in which every peer can communicate with every other peer over a single hop. But what happens, if actually traversing another peer's network to reach a third is unavoidable, because a direct link is physically not possible?

Consider the situation in Figure 3.8. `D`, `E`, and, `F` are present at IXP 1, but `G` is not. There is a second IXP which only `F` and `G` connect to. To enable peering between `D` and `F`, we can again add another peering link (dotted, blue), but `G` cannot establish direct peering with `D` or `G`, because they are not connected to a common IXP. Instead, they have to exchange traffic over `F`, which would require traversing two peering links.

3.2.1 Peering via Parent-Child Links

Since peering links are too limited, we have to either replace or supplement them with parent-child links. In Figure 3.9, the peering links have been replaced by parent-child links, with `D` as parent

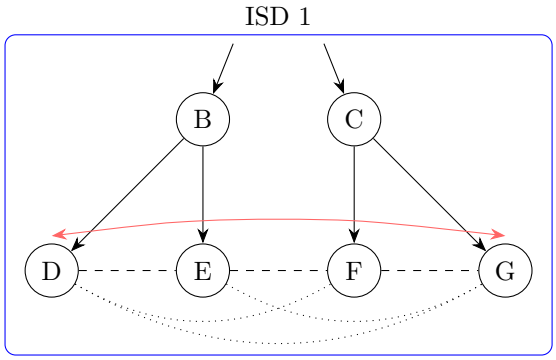


Figure 3.7: The user ASes D and E, E and F, and F and G have peering links (dashed lines). E and F allow their peering partners to use their peering links transitively, resulting in additional peering links (dotted lines).

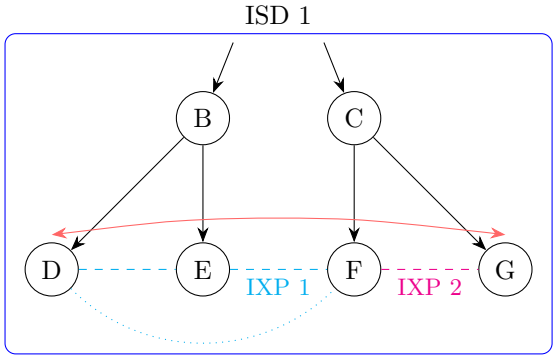


Figure 3.8: D, E, and F are connected to IXP1, F and G to IXP2. Direct peering between D or E and G is no longer an option.

of E, which is parent of F, which is parent of G. With this configuration, beacons travel from D to E to F to G providing up- and down- segments along this path. D can now reach F over the path D-E-F-G. The peering is transitive. However, we have also created other new paths that where not possible before. Take AS E for example. Since F is now its child, F can use E's link to its provider B (blue path), which was not our intention. To eliminate the unwanted paths, the peering ASes have to employ additional policies enforced in another way. Section 3.1.2 list some options.

Parent-child links cannot connect ASes of different ISDs, so the solution of using them as a replacement for peering links breaks down. Figure 3.9 contains a second ISD with the user AS K. Suppose F should be able to reach K on the path F-G-K (green). With the topology in Figure 3.9 that is not possible. No up-segment starting from F passes through G, so the peering link between G and K is unavailable. We can solve this problem by inverting the direction of the parent-child links, putting G at the highest point in ISD 1, and K at the highest point in ISD 2. Figure 3.10 depicts this configuration. Additionally, we have added another AS L as child of K. In this topology, the forwarding path D-E-F-G-K-L is usable. D, E, and F can use up- or down-segments through G, and L can use an up- or down-segment passing through K, effectively connecting all ASes transitively.

But what would happen when there is a third ISD with an AS connecting to D by an inter-ISD peering link? Then G would have to be an (indirect) child of D, while D would also have to be a child of D, thus creating a cycle in the topology graph. Since intra-ISD beaconing does not allow cycles, it is impossible to satisfy all requirements at once. Furthermore, a forwarding path can only contain a single peering link. Since only peering links can connect non-core ASes of different ISDs, transitive peering across more than two ISDs is not possible.

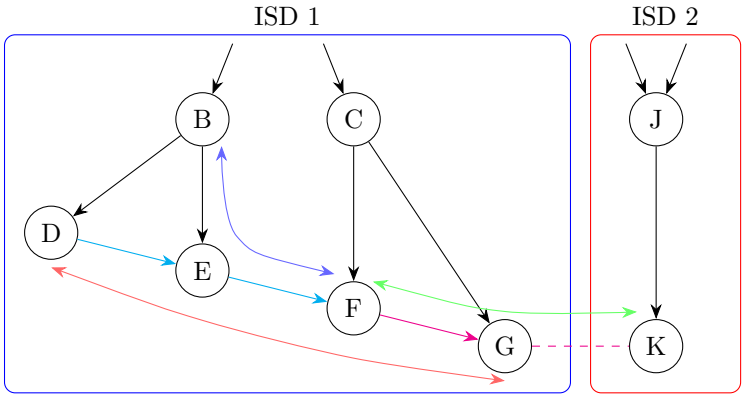


Figure 3.9: Using parent-child links for peering.

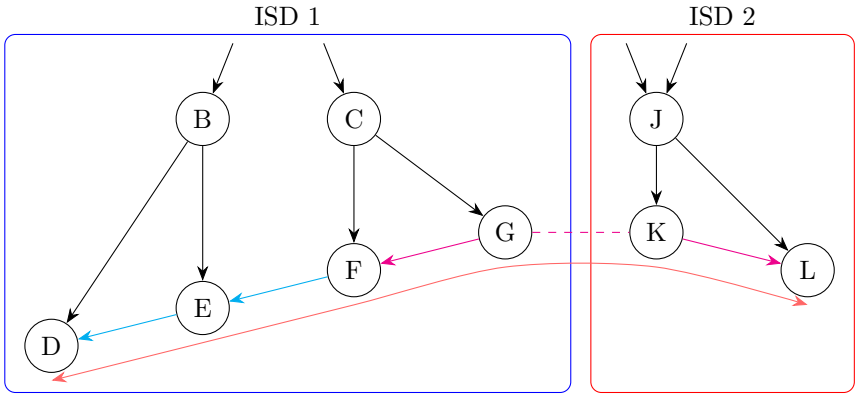


Figure 3.10: Peering using parent-child links is not possible between different ISDs.

3.2.2 Peering with Multiple Identities

One possibility to circumvent the no cycles restriction is to allow the same AS to appear at multiple places in the topology with different AS numbers. SCION end hosts would then not only choose from a selection of possible paths between one source and one destination AS, but from all paths between all identities of the source AS and all identities of the destination AS. Take Figure 3.11 as example. In that topology, `D`, `E`, `F`, and `G` are all peering with each other. `E` offers its peers transit to its other peers, i.e., there should be two hop paths with `E` as middle AS, like `D-E-F`. This is achieved by making `D`, `F`, and `G` children of `E` under different identities than they use to connect to their providers `B` and `C`. If, for example, an end host in `D` needs a path to `F`, it can choose `D`'s second identity as child of `E`, get an up-segment `D2-E-B`, a down-segment `B-E-F2` and construct the path `D2-E-F2`.

In Figure 3.12, we extend the topology by also allowing transitive peering over AS `G`, which adds `D`, `E`, and `F` as children of `G`. This is not enough though, since now `E` and `G` can be used as hops between `D` and `F` in a path over three peering links. Thus, paths `D-E-G-F` and `D-G-E-F` should exist. To achieve that, we add another layer of ASes. `D` and `F` become children of `G2`, as `D4` and `F4`, enabling paths `D2-E-G2-F4`, `F2-E-G2-D4`, `D4-G2-E-F2`, and `F4-G2-E-D2`. Note that paths like `D2-E-G2-F4` and `F2-E-G2-D4` are distinctive. They are constructed from different up- and down-segments, even though they effectively mean the same to the data plane.

It is clear, that allowing all possible transitive paths leads to a combinatorial explosion, introducing a large number of AS identities and possibly redundant paths. Another issue is what happens if peering policies are updated. A single AS changing its peering could change large parts of the peering topology, demanding reconfiguration of many other peers including creating or removing AS identities.

Finding a way to construct a topology satisfying a set of transitive peering policies and deter-

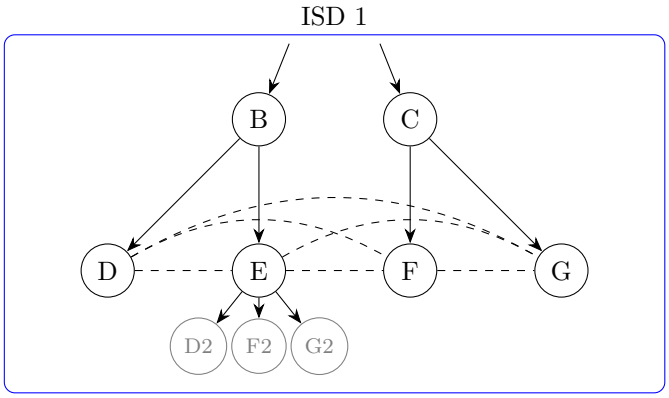


Figure 3.11: `D`, `E`, `F`, and `G` are all peering with each other. `E` provides transitive peering to `D`, `F`, and `G`.

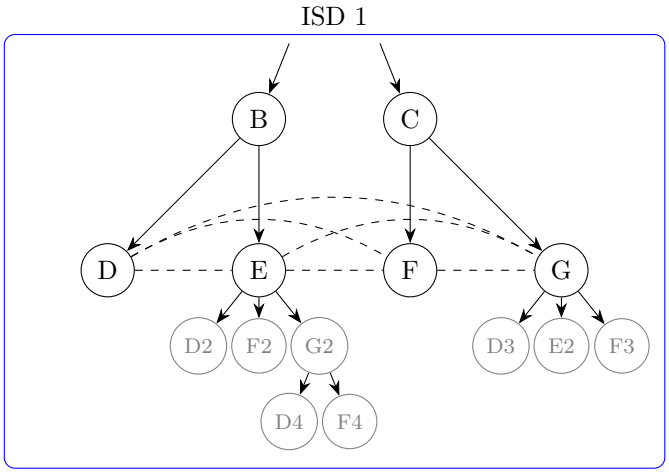


Figure 3.12: In addition to `E`, `G` is providing transitive peering. Therefore, paths traversing three links become available, adding another layer of AS identities.

mining under what circumstances such a topology even exists will be left for future work.

3.3 SCIONLab Coordinator as Peering Coordination Service

An IXP provides its members with L2 connectivity, enabling the connected ASes to engage in bilateral BGP peering sessions. As discussed in Section 2.2.2, multi-lateral peering via a route server adds additional value to an IXP by automating interconnections between all participating ASes.

SCION has no equivalent to a BGP route server. There are no prefix announcements that could be distributed to many ASes at once. Instead, a SCION network topology is defined by core, parent-child, and peering links in an explicit graph. All link types connect exactly two ASes, so peering is always a bilateral affair.

The main advantage of participating at a route server is reduced management and configuration effort. In SCION, a service automating the creation of peering links between ASes at the same IXP could offer similar benefits. The SCIONLab coordinator manages infrastructure AS configuration deployment and user AS creation in the current SCIONLab research network. The coordinator appears as the perfect place to implement peering session management. It already has a global view on the network's links and generates the AS configuration files realizing the topology. Therefore,

we chose to extend to coordinator with a peering coordination service. In the following, we will formulate possible requirements for the SCION peering coordination service, explain the subset of requirements we selected to implement, and discuss how SCION ASes can set up basic peering strategies in the resulting coordinator.

3.3.1 Requirements

From the comparison of BGP and SCION peering in Section 3.1, we derive the following requirements for a SCION peering coordinator:

1. **Augment manual link management with policy based automatic links.**
   The SCIONLab coordinator provides a web-based interface in which links are added or removed by administrators. As a peering manager, the coordinator should be able to automatically generate links based on AS level policies set by AS owners.

2. **Peering policies should cover basic peering strategies.**
   The policies understood by the coordinator should cover the needs of generally open, selective, and restricitve peering strategies. AS owners should be able to specify whether they are open to peer with all new ASes, or only with selected partners. Peering brokering by the coordinator should be opt-in, so ASes with restrictive policies can continue peering over links managed by SCIONLab administrators.

3. **Provide a policy configuration interface to AS owners.**
   There should be an interface for SCIONLab AS owners to configure the peering policies of their own ASes.

4. **Should be able to create core, peering, and parent-child links**
   As we have seen in Section 3.1 and will further elaborate on in Section 3.2, peering in SCION can require all three links types. Thus, the peering coordinator should be able to work with all three types. It might also have to manage multiple identities for the same AS.

For the prototype peering coordinator described in this work, we simplify the above requirements in the following ways:

1. **The coordinator manages peering links only for user ASes.**
   User ASes are already linked to SCIONLab user accounts and can be created by regular users without involving an adminstrator. They allow the gerneral public to connect to and experiment in a large-scale SCION network. Joining as an infrastructure AS and configuring its links is not automated and requires contacting the SCIONLab team. Therefore, starting with automating peering between user ASes seems appropriate for our prototype. At a later stage the prototype could be extended to support infrastructure ASes as well.

2. **Core AS peering is not supported.**
   Since user ASes are always non-core ASes, there is no need to automatically establish links between core ASes or core and non-core ASes at this stage. Manual core link configuration by an administrator is still possible, of course.

3. **Transitive peering via parent-child links is not supported.**
   We do not support transitive peering yet, which would require creating parent-child links as a substitute to peering links. Introducing a transitive peering link in an existing topology requires more complicated global re-configuration. Section 3.2 presents some ideas on how transitive peering could be achieved, but we have not implemented them.

3.3.2 Peering Coordinator Policies

Our peering coordinator allows registered users to specify peering rules for every user AS they own. Rules are specified for every user AS and IXP it connects to individually. There are two kinds of rules: AS policies and ISD policies. AS policies specify with which individual other user ASes peering is allowed, and ISD rules do so for sets of user ASes based on ISD membership. The

rules follow a simple white- and blacklist pattern, either allowing or denying peering. A peering links is only established if both ASes allow peering with each other.

Example AS policies are listed in Table 3.1. In this example, AS `D` denies peering with `F`, effectively blacklisting it. `D` allows peering with `K`, whitelisting it and requesting a peering link. A peering link between `D` and `K` is established, if `K` also whitelists `F`. A peering link between `D` and `F` is never created, because `D` forbids peering with `F`.

Table 3.2 list some ISD based policies. In this example, AS `D` requests peering with all other ASes in ISD 1, whereas `K` denies peering with all ASes in its own ISD, ISD 2. Note that an AS can disallow peering with an ISD, but still whitelist selected ASes from the blacklisted ISD to peer with them. Conversely, some ASes from a whitelisted ISD can be excluded from peering by blacklisting them individually. Simultaneously white- *and* blacklisting a certain AS or ISD is not allowed.

Policy Resoultion Algorithm A link between two ASes `A` and `B` is established, if `B` is in the set of ASes `A` allows peering with and `A` is in the set `B` allows peering with. We call the set of ASes `A` wants to peer with `A`'s peering candidate set. The candidate set can be different for different IXPs. Algorithm 1 describes how a candidate set is constructed.

Algorithm 1 Get the set of ASes *as* is interested in peering with at *ixp*.

```
function GET_CANDIDATE_SET(as, ixp)
    candidate_set = {}
    for isd in ISDs whitelisted by AS as (at IXP ixp) do
        Add all ASes (except a itself) in isd to the candidate set.
    end for
    for peer_as in ASes blacklisted by AS as (at IXP ixp) do
        Remove peer_as from the candidate set.
    end for
    for peer_as in ASes whitelisted by AS as (at IXP ixp) do
        Add peer_as to the candidate set.
    end for
    return candidate_set
end function
```

Note that blacklisting an ISD has no effect on the candidate set, since disallowing peering is the default if there is no other rule. The coordinator still allows for blacklisting ISDs to maintain symmetry with AS specific policies. Blacklisting an ISD also prevents the later accidental addition of a rules whitelisting the same ISD, because accept and deny rules for the same ISD are not permitted in the policy set at the same time.

Example Returning to the example policies in Table 3.1 and Table 3.2, we can now determine between which ASes peering links should be established. Figure 3.13 shows an incomplete topology with user ASes `D`, `E`, `F`, `G`, `K`, and `L`. If we were to apply just the first four (orange) ISD rules from Table 3.2, we would have all possible links between `D`, `E`, `F`, and `G`, but because the first policy in Table 3.1 forbids peering between `D` and `F`, the link between these ASes is missing.

IXP	AS	Peer AS	Accept/Deny
IXP 1	D	F	Deny
IXP 1	D	K	Accept
IXP 1	E	K	Accept
IXP 1	G	L	Accept
IXP 1	L	G	Accept
IXP 1	K	L	Accept
IXP 1	L	K	Accept

Table 3.1: AS based peering policies.

The links `D-K` and `E-K` are created because of the green rules, with which `D` and `E` request peering with `K`. Since `K` additionally wishes to peer with as many ASes in ISD 1 as possible, the links are established. There are no links between `F` or `G` and `K`, because neither of them allow peering with `K` specifically or ISD 2 as a whole.

IXP	AS	Peer ISD	Accept/Deny
IXP 1	D	ISD 1	Accept
IXP 1	E	ISD 1	Accept
IXP 1	F	ISD 1	Accept
IXP 1	G	ISD 1	Accept
IXP 1	K	ISD 1	Accept
IXP 1	K	ISD 2	Deny

Table 3.2: ISD based peering policies.

The purple link between `G` and `L` illustrates a peering relation caused just by AS rules. `G` wants to peer with `L` and vice versa, so the link is established. Finally, the blue link between `K` and `L` exists because the AS policies allowing `K` and `L` to peer take precedence over the `K`'s policy to generally not peer with ASes in ISD 2.

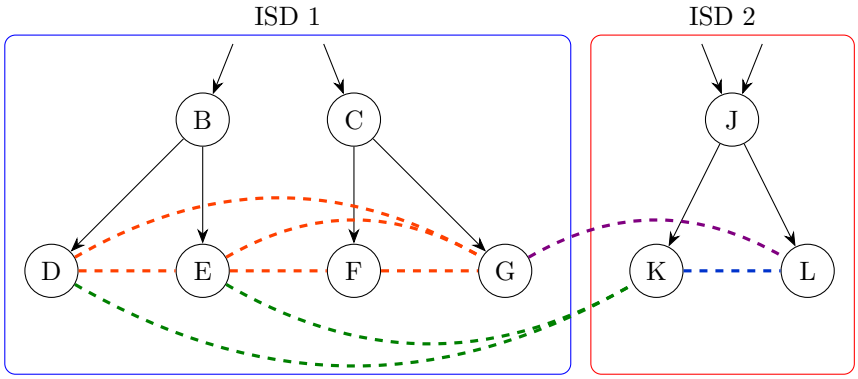


Figure 3.13: Example topology with attachment points `B`, `C`, and `J`. The leaf ASes `D` to `G`, `K`, and `L` are user ASes. The peering links resulting from the policies in Table 3.1 and Table 3.2 are denoted by dashed lines.

3.3.3 Expressing Open, Selective, and Restrictive Peering Inclination

The simple AS and ISD white- and blacklisting approach can express generally open, more selective and restrictive peering policies. An AS with a generally **open peering policy** can whitelist whole ISDs to potentially peer with everyone in an Internet region. Excluding individual ASes from the set of peers is still possible with deny-AS rules.

ASes with a more **selective policy** can use accept-AS rules to peer with ASes conforming to their policies. They might also want to allow peering with selected ISDs, if these ISDs meet their criterions as a whole.

ASes with **restrictive policies** are generally not interested in participating in an automatic peering system. Since specifying no peering rules in the coordinator is equivalent to blacklisting everyone, user AS owners not interested in the peering coordinator do not have to change anything. Alternatively, they could blacklist all ISDs to document their disinterest in peering with other user ASes.

3.4 Deploying SCION in the Data Plane

Deploying SCION on an IXP does not require any changes to the IXP's infrastructure. At the moment, SCION is implemented as an overlay protocol over IP/UDP. For native SCION links, the IP/UDP wrapper is dropped, and SCION is placed directly above the L2 protocol in the protocol stack. As long as the IXP provides L2 connectivity, IXP members can talk over SCION. However, many IXPs offer more than just simple interconnection, for example route servers and DDoS protection in the form of blackholing. We have already discussed how the SCION equivalent of a route server could look like in the previous section. In this section, we are going to explore reasons to integrate SCION in an IXP's data plane and sketch how this can be done efficiently, even though no SCION aware network hardware exists yet.

3.4.1 Reasons for IXP Participation in the SCION Data Plane

IXPs do not participate in IP routing, because the enormous bandwidth they handle would overload any IP router. Instead, they essentially provide a large Ethernet switch connecting all participants together, forwarding packets according to MAC addresses. They do not appear as a hop from an IP perspective. In SCION, however, the IXP participating more actively in the data plane is possible and might even be desireable.

In SCION, all routing decisions are made ahead of time, the packet header contains the entire AS-level path a packet is supposed to take (*packet-carried forwarding state*). Everything intermediate border routers have to do is to forward packets to the destination port given in the currently active hop field. This is no more complex than switching according to MAC addresses.

Alternative Peering Topology The IXP appearing as a hop in the forwarding path enables another way to handle IXP peering links in SCION. Figure 3.14 depicts a topology with two IXPs labeled 1 (blue) and 2 (red). Instead of establishing peering links, participating ASes become child ASes of the IXP, which appears as a SCION AS itself. For example, ASes `D` and `E` are connected to IXP 1. They can communicate on the path `D-1-E`.

As a SCION AS, the IXP has to be connected to the ISD core, or be a core AS connecting to other core ASes itself. Of course, ASes connected to the IXP are not supposed to use these links, so they need only minimal bandwidth for control plane traffic. If a core AS whishes to peer at an IXP represented as a SCION AS, the IXP has to run a core AS as well and connect peering core ASes via high-bandwidth core links.

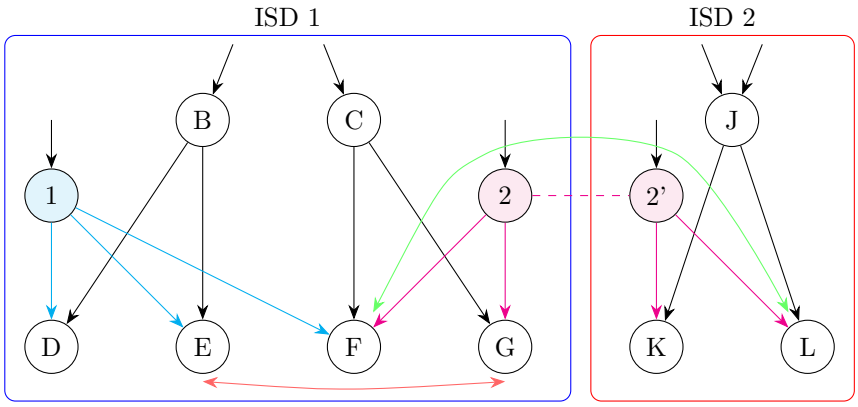


Figure 3.14: IXP as parent AS.

Peering between different ASes from different ISDs is achieved by having an IXP AS in every ISD peers are in. For example, IXP 2 has two ASes `2` and `2'` in ISD 1 and ISD 2, respectively. The two ASes are connected via a peering link, enabling ASes from both sided to talk to each other (green path). The transitive peering problem outlined in Section 3.2 is not solved by this topology, however. `F` cannot provide transit from IXP 1 to IXP 2 for `E` and `G` (red path).

Visibility of the IXP to End Hosts It might be more in tune with SCIONs philosophy to give end hosts control over the forwarding path to have IXPs as explicit hops. This way, end hosts can more easily filter out routes crossing IXPs who they deem untrustworthy. Excluding such paths is still possible if the IXP does not show up as a SCION AS, but requires the end hosts to know which peering links use which IXP, which is much less transparent.

DDoS Defence Currently, some IXPs offer blackholing as a mechanism to mitigate DDoS attacks (see Section 1.1 and Section 2.2.3). Blackholing discards attack traffic in the IXP's switch fabric when certain conditions are met. SCION's own DDoS defence mechanisms rely on the border router's ability to cryptographically verify hop fields. If the source of an attack does not have a valid hop field, the attack traffic is discarded before it can enter the AS, but only after it has reached the AS's border routers. Here an IXP could provide additional value by already verifying the hop field of packets when the enter the IXP. This would allow dropping unwanted traffic before it can congest an AS's port at the IXP.

3.4.2 Bare-Metal Switches as SCION Routers

Some bare-metal switches have the ability to match user defined header fields. This feature could be used to match hop fields in the SCION packet header. Hop fields are usually verified via symmetric cryptography that is not available in bare-metal switches, but because hop fields representing the same hop stay the same in every packet, verification could be done in software once and after that by simply comparing hop fields of arriving packets to valid hop fields in TCAM. This way, the bare-metal switch could efficiently determine whether a packet is valid and to which port to forward it. It could not perform all steps required of a full SCION border router, but at least some work would be offloaded to dedicated hardware.

We already mentioned to difficulty of extracting the right hop-field from the SCION header with a bare-metal switch in the introduction (Section 1.1). In short, the problem is that the location of the hop-field relative to the well-known packet headers (Ethernet, IP, UDP) must be constant and near the start of the SCION header. Unfortunately, the location of the hop-field we would be interested in varies widely and can be so far back in the packet, that a bare-metal switch might no be able to access it at all. Therefore, it would be necessary to prepare packets meant to be interpreted by bare-metal switches in a special way.

Changes planned to the SCION header at the time of this writing might offer an opportunity to incorporate a special packet format, catering to bare-metal switches, into SCION. The new SCION header format keeps the basic structure of a common header, followed by variable length addresses and the variable length forwarding path of the current SCION header (compare Figure 1.1), but the internal structure of these header sections is changed. Important for our use case is that the common header now contains a *PathType* field. The path type is a 8 bit value determining the format of the forwarding path. A standard SCION path will have type 0. Other path formats allow the SCION extensions EPIC (path type 1) and COLIBRI (path type 2) to be natively supported without resorting to extension headers. EPIC and COLIBRI will add optional headers between the address header and the regular forwarding path following the format of path type 0 (see Figure 3.15).

Common header *PathType, ...*	8 bytes
Addresses *IPv4/IPv6 source and destination, AS, ...*	24-48 bytes
EPIC/COLIBRI header *optional*	variable length
Forwarding path *InfoFields, HopFields, ...*	variable length

Figure 3.15: New SCION header with optional EPIC/COLIBRI header.

By introducing another path type, say BARE-METAL, we could add a special header, to be read by bare-metal switches, after the address header. As with EPIC and COLIBRI, the format of the rest of the forwarding path would stay the same as with path type 0 (see Figure 3.16).

Common header *PathType, ...*	8 bytes
Addresses *IPv4/IPv6 source and destination, AS, ...*	24-48 bytes
BARE-METAL header *current HopField and current InfoField*	20 bytes
Forwarding path *InfoFields, HopFields, ...*	variable length

Figure 3.16: New SCION header with proposed bare-metal switch header.

A possible layout for the bare-metal header is given in Figure 3.17. It simply consists of copies of the current hop and the current info field. In the new SCION header, hop fields are standardized to a length of 12 bytes. The new info fields have a length of 8 bytes, yielding a fixed total length of 20 bytes. For hop field verification via TCAM matching including the current hop field would be sufficient. We also include the current info field, because verifying the hop field cryptographically requirers the value of the `SegId` field contained in the info field. The reason to include the hop field before the info field is to keep the most important information as close to the start of the packet as possible.

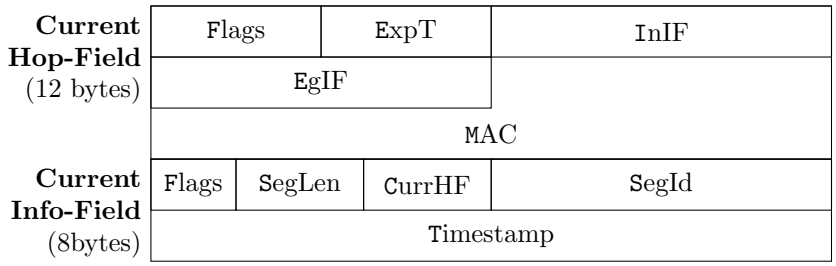


Figure 3.17: Bare-metal switch header containing the current hop and info field.

One problem remains with the header structure in Figure 3.16: The address header is still of variable length, so we have not achieved our goal of keeping the current hop filed at a constant position. Figure 3.18 depicts the layout of the address header. As we can see, the SCION parts of the destination and source address have a fixed length of 8 bytes each, but the host addresses within the source and destination AS depend on the protocols these ASes employ internally.

The best we can hope for is that most packets will use the same host address length. For example, 4 bytes for source and destination each with IPv4, or 16 bytes each with IPv6. If there is only a small number of different possible positions the hop field in the bare-metal header can occupy, we could extract 12 byte blocks from all these positions and match them against know hop fields in TCAM.

If the new path type we proposed would be implemented, IXPs could offer a service where SCION packets are verified and forwarded according to the SCION header. To use this service, border routers in participating ASes have to prepare packets in a special way: They change to path type to BARE-METAL, and add the bare-metal header between the *address* and *forwarding path* fields. The bare-metal header contains the hop and info field current, when the packet leaves the last border router before traversing the IXP. These fields must be verified by the border router of the next hop AS. The bare-metal switch would perform this verification within the IXP, so that invalid packets never reach the next hop AS and cannot congest the next hop's IXP port. Valid packets reaching the next hop AS, would be stripped of the bare-metal header and their path type reverted to the standard SCION type, before they are processed as usual.

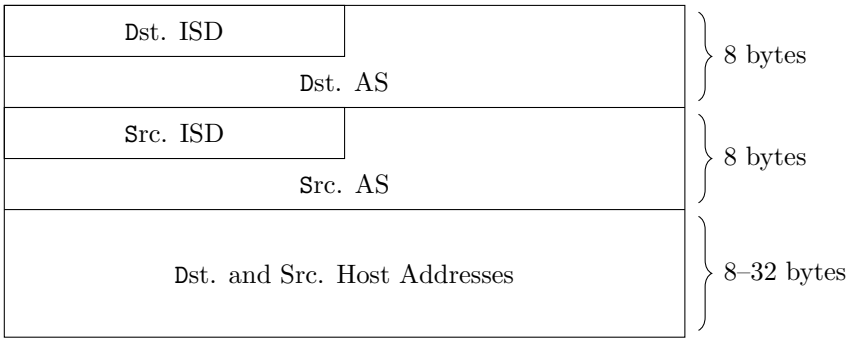


Figure 3.18: SCION address header.

The only missing ingredient in the process just described is how the bare-metal switch is supposed to learn which hop fields are valid. Verifying a hop field cryptographically requirers a symmetric key only the original issuer of the hop field has access to. Thus, allowing the IXP to verify hop fields entails either trusting the IXP with the hop field verification key, or providing the IXP with a list of valid hop-fields via some kind of API.

A limitation of the proposed bare-metal header and the matching of hop fields against a TCAM table is that they will only work with standard SCION packets. SCION extensions like COLIBRI are not supported. For one, because EPIC and COLIBRI already use their own path type, but more importantly because the hop fields change with every packet, prohibiting the TCAM matching approach. Nevertheless, bare-metal switches might be able to fill the gap between hardware implementations of SCION, which do not exists yet, and fully programmable switches, which are costly and hard to use.

Chapter 4

Testbed and Coordinator Implementation

To evaluate how SCION behaves when a large number of peering connections are dynamically added or removed we need a way to efficiently set up and modify a large SCION topology. In Section 4.1, we introduce a software framework we created for that purpose. Subsequently, we describe the modifications we made to the SCIONLab coordinator in order to support the peering policies discussed in Section 3.3.

4.1 Testbed Framework

We developed a software framework to facilitate experimenting with SCION in an IXP like environment. Our main concern was to enable efficient set up and modification of large SCION topologies spanning many host computers. To this end, we created a wrapper around the experimentation and development tools the current SCION implementation provides. Thereby simplifying the experiments we conducted for the previous section and automating the evaluation we present in the next chapter. Our testbed software is written in Python. The source code is available online[1].

The open-source SCION distribution provides a script for creating local standalone SCION networks. The script takes a YAML file defining the ASes and links of the topology as input and produces the configuration files for all requested ASes. The SCION services of the topology are then started through supervisord[2], a process monitoring system. Installation instructions for SCION and example topology definitions are available in the SCION source repository[3].

Using the local topologies via the setup script was not sufficient for our purposes, because the generated topologies are meant to run on a single host computer. We needed a solution to automate deployment on multiple host to enable larger evaluations. Additionally, modifying the SCIONLab coordinator to support peering session management is one of the main goals of this thesis. ASes running in a topology managed by an instance of the coordinator download their configuration from there. Therefore, we needed a different way to set up our SCION networks. To maintain compatibility with the existing scripts, we decided to extend the YAML schema of the topology files.

The SCION repository also provides a script[4] for building Docker containers with SCION and all necessary dependencies. By running SCION in containers, we can easily distribute ASes over multiple hosts. It also allows us to isolate the network traffic of certain SCION links and force it through physical network interfaces, even when both ends of the connection lie on the same host.

We decided to run each SCION AS with all its services in its own container. With Docker, one would normally split each individual service in its own containers, but we did not need this level of granularity. We simply use Docker as a convenient container virtualization system.

[1] `https://github.com/lschulz/scion-ixp-testbed`
[2] `http://supervisord.org/`
[3] `https://github.com/netsec-ethz/scion`
[4] `https://github.com/netsec-ethz/scion/blob/scionlab/docker.sh`

Overview There are three main "modes" our testbed software can operate in. First, it can create a standalone topology, without a coordinator, running locally. For this purpose, it invokes the usual topology script from the SCION distribution and post-processes the resulting configuration before setting up the containers. In the second mode, a container with the SCIONLab coordinator is instantiated in addition to the SCION ASes. The coordinator's database is initialized with the user supplied topology specification, on which the configuration subsequently downloaded by the ASes is based. The third mode extends coordinator managed topologies to span multiple hosts.

In Section 4.1.1, we provide more details on how containers are interconnected. Section 4.1.1, Section 4.1.2, and Section 4.1.4 explain the three modes of operation in more detail and give configuration examples.

4.1.1 AS Container Networking

We have implemented four ways to interconnect AS containers. Containers running on the same host can be connected via Docker bridges[5] or via Open vSwitch[6] (OVS). Containers on the same or different host are connected by Docker overlay networks[7]. If the containers are running on two different host, port forwarding can be configured to connect them directly, without the overhead of Docker's overlay networking. Refer to Table 4.1 for an overview.

Network Type	Identifier	Local or Remote ASes	Maximum Size
Docker Bridge	`docker_bridge`	local	unlimited
Open vSwitch (OVS)	`ovs_bridge`	local	unlimited
Docker Overlay	`overlay`	local and remote	< 254 ASes
Port Forwarding	`host`	remote	unlimited

Table 4.1: Network types supported by the testbed framework and their identifiers in topology definitions.

When connecting containers on the same host, Docker bridges should be preferred, because the overhead of Open vSwitch is likely higher. We support interconnecting over OVS as an alternative, because it understands OpenFlow, enabling it to stand in for an OpenFlow hardware switch during initial development. We expect this to be useful for implementing the bare-metal switch header matching outlined in Section 3.4.2. The limitation of only supporting local connections comes from our framework. We simply did not need to connect different host to OVS yet, but could support it in the future.

Docker overlay networks are mainly used for Docker swarm mode services, but they can also connect standalone containers. To connect a standalone container to an overlay network, the Docker daemon has to be in swarm mode. Therefore, hosts participating in a multi-host topology must first join a common swarm. The documentation accompanying our software includes instructions on how to configure swarm mode.

Unfortunately, Docker overlay networks are limited to /24 subnets[8], allowing only 254 endpoints to connect. Since every Docker host participating in the overlay network gets an IP address in its subnet, the number of IPs available to SCION ASes is reduced further. In the future, it would be possible to support larger topologies by configuring multiple IP subnets per overlay network[9], but we have not implemented this feature yet.

Our primary reason for supporting forwarding AS traffic to the host's network interfaces is to enable placing bare-metal switches or other suitable hardware between ASes. This way, our testbed simplifies experimenting with hardware support for SCION. Another good reason to use direct networking instead of Docker's overlay networks might be better performance[27].

Configuration Example Listing 4.1 contains a section of a topology configuration file, as understood by our framework. It defines four network, one of each type. Every network must define

[5] `https://docs.docker.com/network/bridge/`
[6] `https://www.openvswitch.org/`
[7] `https://docs.docker.com/network/overlay/`
[8] `https://github.com/moby/moby/issues/30820`
[9] `https://docs.docker.com/engine/reference/commandline/network_create/`

its type and the IP subnet to use in CIDR notation. IPv6 networks are supported in topologies managed by the coordinator, but not well tested. Consult the IPv6 example (`topologies/coordinator/ipv6.yaml`) in the source distribution for more information. Note that subnets must not overlap each other.

Listing 4.1: Network definitions.

```
networks:
  "network1":
    type: "docker_bridge"
    host: "localhost"
    subnet: "10.1.1.0/24"
  "network2":
    type: "ovs_bridge"
    host: "localhost"
    subnet: "10.1.2.0/24"
  "network3":
    type: "overlay"
    host: "localhost"
    subnet: "10.1.3.0/24"
  "network4":
    type: "host"
    subnet: "192.168.244.0/24"
```

Bridge and overlay networks also have to specify the host they should be created on. For bridge networks, the host is the Docker host that will run the containers connecting to that bridge. In case of an overlay network, it is the host the network is initially created on. This host must be a Docker swarm manager node. The network is expanded to the other hosts by Docker, when their containers first join the network.

Valid options for the host filed are `localhost`, identifying the local computer, and any host name defined in the hosts section of the topology file (see 4.1.4). The default value, if the host field is missing, is `localhost`.

4.1.2 Standalone Topologies

Standalone topologies are generated by invoking the `scion.sh topology` command from SCION source repository. For our testbed framework, we have extended the topology files this command expects. Listing 4.2 defines a topology in our extended format. We have already introduced the `networks` section, declaring the networks which connect the Docker containers. In this example, we have a Docker bridge, and an OVS bridge network.

The example topology contains three ASes, one core AS and two non-core ASes connected via parent-child links. Moreover, there is a peering link between the two non-core ASes. In the `links` section, we find the next extension. In addition to the endpoints and type, links can define over which network they are established. The network can either be a network name defined in the `networks` section or the name of an IXP defined in the `IXPs` section, which we will discuss shortly. As shortcut for declaring a Docker bridge, the network can be given as an IP subnet directly. If a link does not declare a network explicitly, a new Docker bridge is created for this link. The subnet of the bridge is allocated from the subnet given in the `link_subnet` field of the `defaults` section. If no default link subnet is given, omitting the network on a link is an error.

Link endpoints specifications follow the format understood by `scion.sh topology`. They consist of an ISD-AS identifier followed by an optional border router name, separated by a hyphen. This is followed by an optional numeric interface ID, separated by a # character. If no border router name is given, a new border router is created for every endpoint. Interface IDs start at 1 and must be unique across the entire AS.

Since our goal is to simulate an IXP, we have also introduced IXPs as concept in the topology specification. IXPs are defined in the section of the same name. At the moment, IXPs only declare the network they will be using.

Running the Topology A topology is created from a specification file by running

Listing 4.2: Standalone topology.

```
defaults:
  subnet: "127.0.0.0/8"
  link_subnet: "10.1.1.0/24"
  zookeepers: {1: {addr: 127.0.0.1}}
networks:
  "infrastructure":
    type: "docker_bridge"
    subnet: "10.1.2.0/24"
  "ixp_network":
    type: "ovs_bridge"
    subnet: "10.1.3.0/24"
IXPs:
  "ixp1":
    network: "ixp_network"
ASes:
  "1-ff00:0:a": {core: true}
  "1-ff00:0:b": {cert_issuer: "1-ff00:0:110"}
  "1-ff00:0:c": {cert_issuer: "1-ff00:0:110"}
links:
  - {a: "1-ff00:0:a-br1", b: "1-ff00:0:b#1",
     linkAtoB: CHILD, network: "10.1.4.0/24"}
  - {a: "1-ff00:0:b-br1", b: "1-ff00:0:c#1",
     linkAtoB: CHILD, network: "infrastructure"}
  - {a: "1-ff00:0:b-peer#2", b: "1-ff00:0:c-peer#2",
     linkAtoB: PEER, network: "ixp1"}
CAs:
  CA1-1: {ISD: 1, commonName: CA1-1}
```

```
$ ./ixp-testbed.py topology example.yaml
```

in the testbed framework's root directory. The command creates a directory `network-gen` containing the generated files. We refer to this directory as the topology's work directory. A different work directory can be specified with the `-w` option, for example to run multiple topologies at the same time. In this case, it is also advisable to name the topology with the `-n` option, like this:

```
$ ./ixp-testbed.py -w /.../example topology example.yaml -n example
```

The topology name is used as prefix for Docker containers and networks to avoid name collisions. The topology is started by

```
$ ./ixp-testbed.py start
```

and stopped by

```
$ ./ixp-testbed.py cntrs stop
```

It is also possible to just launch and connect the AS Docker containers without starting SCION yet, and to terminate the SCION ASes without cleaning up the containers for faster restarts. Refer to the documentation in our source repository for more information.

The configuration and cache files, as well as logs of ASes running on the local computer are mapped from host directories, named for the AS owning them, in the work directory. For a standalone topology, these directories are created by splitting up the configuration produced by `scion.sh topology`. Another noteworthy file in the work directory is the log file, simply called `log`.

IXP Link Experimentation We provide a few functions to add, remove, and modify links over an IXP in a running network. The general syntax is as follows:

```
$ ./ixp-testbed.py link [add|remove|modify] <IXP> <AS1> <AS2> --type <link type>
```

With this command, peering, but also parent-child and core links established over the given IXP can be manipulated. Note that the effected ASes are restarted when the topology is running at the time the command is issued. If a new link is added and any of the involved ASes are not already connected to the IXP's network, the connection is made. Conversely, ASes no longer using a certain IXP are disconnected from the IXP's network.

4.1.3 Coordinator Managed Topologies

To test and evaluate our peering extensions to the SCIONLab coordinator, our testbed must support setting up topologies including a coordinator instance. If a topology contains a coordinator, the `scion.sh topology` command is not used. Instead, the topology definition is transformed to a Python script. This script, when executed in the Python context of the coordinator, initializes the coordinator's database with the requested topology. When the coordinator is running, it generates the AS configuration files, which are fetched and installed by the ASes.

Just as the SCION ASes, we run the coordinator in a Docker container. The coordinator container is connected to every AS container, so that ASes can access the coordinator's API. Listing 4.3 shows a configuration for the same topology as Listing 4.2, but this time with a coordinator. The coordinator container, and the coordinator itself are configured in the `coordinator` section.

Listing 4.3: Coordinator-managed topology.

```
networks:
  "coord":
    type: "docker_bridge"
    subnet: "10.1.1.0/24"
  "infrastructure":
    type: "docker_bridge"
    subnet: "10.1.2.0/24"
  "ixp_network":
    type: "ovs_bridge"
    subnet: "10.1.3.0/24"
coordinator:
  network: "coord"
  expose: "8000"
  expose_on: "192.168.244.4"
  users:
    "admin":
      email: "admin@example.com"
      password: "admin"
      superuser: true
    "user1":
      email: "user1@example.com"
      password: "user1"
IXPs:
  "ixp1":
    network: "ixp_network"
ASes:
  "1-ff00:0:a": {core: true, attachment_point: true}
  "1-ff00:0:b": {owner: "user1", ixps: ["ixp1"]}
  "1-ff00:0:c": {owner: "user1", ixps: ["ixp1"]}
links:
  - {a: "1-ff00:0:a-br1", b: "1-ff00:0:b#1",
     linkAtoB: CHILD, network: "10.1.4.0/24"}
  - {a: "1-ff00:0:b-br1", b: "1-ff00:0:c#1",
     linkAtoB: CHILD, network: "infrastructure"}
```

The management network, connecting the coordinator to the AS containers, is given in the `network` field. The `expose` field specifies a TCP port on the Docker host at which the coordinator is made available to the outside world. `expose_on` specifies an address to bind the coordinator to on the host. `expose` and `expose_on` are passed to Docker, which takes care of configuring the forwarding. If `expose_on` is omitted, the coordinator is made available on all host interfaces.

User accounts are declared in a `users` section. Every user must have an email address, which is the login name, and a password. At least one administrator account should be set up initially, to make the admin web-interface accessible. As described in Section 2.5, the coordinator distinguishes between "infrastructure" and "user" ASes. User ASes have an owner, and infrastructure ASes do not. ASes owned by administrator account are user ASes, just as ASes owned by a regular user. The owner of an AS is annotated in the AS declarations as `owner`. Recall that user ASes must attach to exactly one attachment point AS. Attachment points are annotated with `attachment_point: true`. Every user AS must have a link to an attachment point in the `links` section. Note that user ASes can not be core ASes or attachment points themselves.

In the regular SCIONLab coordinator, user ASes cannot have additional links besides the one to their attachment point. For our framework, we made a minor modification to allow adding additional links to user ASes in the topology file and the administrator interface. This way, we were able to implement the peering policy model from Section 3.3. As a side effect parent-child links between user ASes also become available. That means, that our peering policies are not limited to leaf ASes, even though we limit them to user ASes.

To facilitate peering between user ASes, the coordinator must know which user ASes are members of which IXP. IXP membership is declared as a list of IXP names in the user ASes. Note that we have omitted the peering link between AS `ff00:0:b` and `ff00:0:b`. This is because the coordinator will create peering links between user ASes based on their peering policies. The configuration of peering policies is described in Section 4.2. If IXP links are specified in the topology file, they are static links not effected by policies.

The Coordintor Runs in Debug Mode It is important to note that our testbed runs the coordinator in debug mode with Django's built-in test server and an SQLite database. Therefore, the testbed coordinator should not be made accessible from the Internet. If a Docker overlay network between multiple hosts is used as management network, it might be advisable to enable encryption on that network. Add `encrypted: true` to the network declaration to do so.

Besides security, running the coordinator in debug mode has two more drawbacks. The first is reduced performance. Running the coordinator in a fully deployed configuration with a more fully featured database management system would likely result in better performance. However, we are not evaluating the performance of the coordinator itself, but rather of the core SCION services. Futhermore, we did not encounter any obvious performance bottlenecks in the coordinator.

The second drawback, is that automatic deployment of AS configuration updates is not working in debug mode. In the SCIONLab network, the coordinator has SSH access to the attachment point ASes. When user ASes are created, deleted, or modified, the coordinator can push the configuration updates to the effected attachment point. Our testbed does not support adding or removing ASes during runtime at the moment, so missing automatic reconfiguration of the attachment points is not critical. However, the feature of initiating a configuration update from the coordinator's side could be useful to push the effects of policy changes to user ASes.

Our current solution to trigger AS configuration updates is issuing an `update` command to the testbed script, which then initiates configuration updates in the AS containers. Since the testbed already configures SSH servers in all ASes and sets up access for the coordinator, it should be straight forward to support a more automatic approach with a fully deployed coordinator in the future.

4.1.4 Multi-Host Topologies

The final mode supported by our software deploys containers on multiple hosts. Multi-host mode always requirers a coordinator to configure the ASes. To enable multi-host mode, the topology specification assigns ASes and the coordinator to remote hosts declared in a `hosts` section. In Listing 4.4, the coordinator and AS `ff00:0:c` run on a host named `remote_host`. The remaining ASes use the default host specification of `localhost`. `localhost` always refers to the local Docker host and cannot be redefined.

A remote host joining the topology must fullfil the following requirements: (1) It must run the Docker daemon. (2) The local computer must be able to connect to the remote host via SSH.

Docker images for ASes and the coordinator are always build locally. When a topology is started, the images are synchronized with all remote hosts. Therefore, only the local machine requires a SCION installation. The configuration and logs of local ASes are mapped from the topology's work directory like in the other testbed modes. In remote ASes, the data is available within the container only.

The addresses of the remote SSH servers must be specified as `ssh_host` for every host defined in the `hosts` section with the exception of `localhost`, of course. Non-standard SSH ports are set as `ssh_port`. The user name with which to log in is given by `username`.

Internally, two different libraries are used for SSH connections. To execute commands directly on the host and copy files, we use `ssh2-python` which provides Python bindings to `libssh2`. For

bulk operations across multiple hosts, we use the library `parallel-ssh`, which by default uses `ssh2-python` as backend. For communication with the remote Docker daemon however, we rely on Docker's client-server architecture. The Docker server can be bound to a TCP port to allow remote access, but recent versions of Docker enable an even simpler configuration, where using existing SSH access is possible. The Docker SDK for Python[10] uses an older SSH library, called Paramiko, for this purpose.

For authentication with `libssh2`, we provide to option to specify the path to a private key file in the field `identity_file` under the affected host. If no key file is specified, the user's SSH agent is queried for authentication. Password based authentication is not supported. There is no way to pass an explicit private key to Paramiko through the Docker SDK, so running `ssh-agent` with the requisite keys is necessary regardless of whether `identity_file` is given.

Listing 4.4: SCION topology spanning multiple hosts.

```
networks:
  "coord": {type: "docker_bridge", subnet: "10.1.1.0/24"}
  "infrastructure": {type: "docker_bridge", subnet: "10.1.2.0/24"}
  "ixp_network": {type: "ovs_bridge", subnet: "10.1.3.0/24"}
  "physical_network":
    type: "host"
    subnet: "192.168.244.0/24"
hosts:
  "localhost":
    addresses: {"physical_net" : "192.168.244.4"}
  "remote_host":
    ssh_host: "192.168.244.3"
    username: "remote_user"
    addresses: {"physical_net" : "192.168.244.3"}
coordinator:
  network: "coord"
  host: "remote_host"
  expose: 8000
  expose_on: "192.168.244.3"
  users: {...} # same as in the previous example
IXPs:
  "ixp1":
    network: "ixp_network"
ASes:
  "1-ff00:0:a": {core: true, attachment_point: true}
  "1-ff00:0:b": {owner: "user1", ixps: ["ixp1"]}
  "1-ff00:0:c": {host: "remote_host", owner: "user1",
                 ixps: ["ixp1"]}
links:
  - {a: "1-ff00:0:a-br1", b: "1-ff00:0:b#1",
     linkAtoB: CHILD, network: "infrastructure"}
  - {a: "1-ff00:0:b-br1", b: "1-ff00:0:c#1",
     linkAtoB: CHILD, network: "physical_net"}
```

Host Networks Listing 4.4 also illustrates the use of port forwarding to enable links between ASes on different hosts without resorting to a tunneled connection. In the `networks` section, we define a network `physical_net` of type `host`. Host networks must specify the common subnet of the Docker hosts involved. Additionally, the IP address of every machine participating in the network must be known. The IPs are specified in the `addresses` mapping of every host. Specifying addresses for the local computer is the reason `localhost` can be declared in the `hosts` section.

In our example, ASes `ff00:0:b` and `ff00:0:c` communicate directly over their host's network. Thus enabling processing of SCION traffic between them with network hardware like a bare-metal switch.

The host network type is implemented by requesting Docker to publish ports from inside the AS containers to the host. Unfortunately, Docker only allows setting up published ports during container creation. Therefore, we cannot add more SCION border router interface to a running AS. Thus, host networks are not suitable to be used as IXP networks, were border router interface

[10] `https://docker-py.readthedocs.io/en/stable/`

are created dynamically based on policies. It is still legal to specify a host network in an IXP, but only static links defined in the topology file will work.

It would be possible to to work around the limitation of only publishing ports at container creation by reserving some ports ahead of time. Alternatively, the container could simply be restarted when border router interfaces are added, or the necessary routing table manipulations could be made directly, bypassing Docker. The more interesting concern is, how the testbed controller should be notified of which port forwardings are necessary by the coordinator, or vice versa. A solution to this problem could be implemented in the future, but out purposes did not require it yet.

Network Size Limitations The only network type suitable for the coordinator network in a multi-host topology is `overlay`. Recall from Section 4.1.1 that overlay networks are limited to 254 host. Every AS in the topology has to connect to the coordinator's network. Additionally, every Docker host needs an IP address within the overlay network. Moreover, the coordinator itself needs an IP. Therefore, the maximum network size is limited by $1 + n + m < 254$, where n is the number of ASes and m is the number of hosts.

Docker assigns IP addresses to hosts as soon as a container running on said host is connected to the overlay network. IP addresses are assigned starting from the smallest available without gaps. To simplify address allocation for our AS containers, we reserve a certain amount of IPs for the Docker hosts. By default, we reserve eight IP addresses, consequentially limiting the topology to a maximum of eight host and $254 - 8 - 1 = 245$ ASes. The maximum number of hosts is controlled by the constant `OVERLAY_NETWORK_MAX_HOSTS` and can be increased if necessary.

Supporting larger networks would require implementing a new network type, or extending the maximum size of overlay networks as described in Section 4.1.1.

4.2 Coordinator

We have implemented the peering coordinator described in Section 3.3 by extending the SCIONLab coordinator's database schema with five new models, implementing the simple policy resolution algorithm from Section 3.3.2, and augmenting the coordinator's RESTful API. In the following, we provide some implementation details concerning these contributions.

4.2.1 IXP Peering Models

The database layout in Django is derived from model definitions in Python via an object-relational mapper (ORM). We introduce the concept of an IXP to the coordinator by adding five new Django models. The models and their relations are depicted in Figure 4.1. The model `IXP` represents an IXP, or more precisely a single VLAN at an IXP. If an IXP has multiple VLANs, they would all be described by `IXP` instances. IXPs have an optional label naming them in human readable form. In a topology generated by our testbed program, this label is the name of the IXP in the topology file. For the purposes of referring to an IXP from other models and the peering API a surrogate ID is used. The coordinator also stores the IP subnet (`ip_network`) configured in the IXP's LAN.

IXP members are represented by the model `IXPMember`. Every IXP member has an IP address (`public_ip`) from the IXP's subnet. Additionally, there is an optional `bind_ip` field for ASes behind network address translation (NAT). If the coordinator decides to create a new link based on an IXP member's policy, the public and bind address from the `IXPMember` instance are used to initialize the new border router interfaces. The bind address is the address the border router process binds to. The public address is the address the border router is contacted on by other ASes. We use this distinction to support the `host` network type of our testbed framework (see Section 4.1.1).

`IXPMember` instances refer to hosts instead of ASes directly. Within the framework of the SCIONLab coordinator, an AS has a set of services, like border routers and beacon servers, which can run on different host computers with different IP addresses. Therefore, we need a mapping of IXP IP to hosts instead of to ASes, which might have multiple hosts connecting to different IXPs. However, our prototype only supports peering between user ASes. Since user ASes can only have a

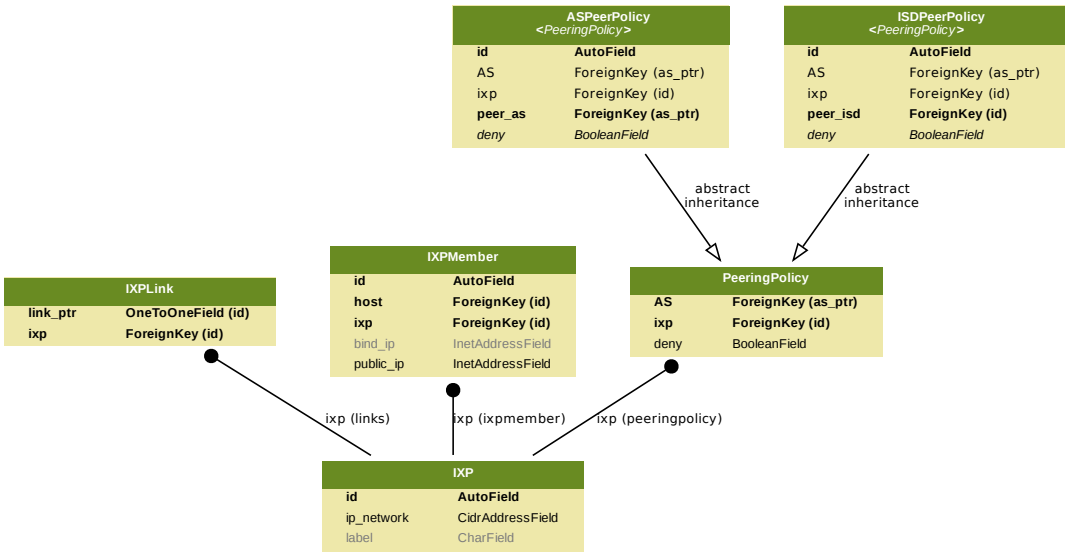


Figure 4.1: Django database models we add to the coordinator.

single host in the current SCIONLab network, there is a one to one mapping between user AS and its host, rendering the distinction largely irrelevant. For the same reason, our policies are defined per AS instead of per host. Nevertheless, supporting per host policies would be a worthwhile extension in the future, for example for peering between infrastructure ASes.

AS and ISD peering policies are represented by `ASPeerPolicy` and `ISDPeerPolicy`, respectively. Both models inherit common fields from the abstract base class `PeeringPolicy`. Their fields are a direct translation of the tables in Section 3.3.2. We store the AS owning the policy (`AS`), the IXP effected by the policy (`ixp`), and the potential peer(s) (`peer_as`, `peer_isd`). A policy is a deny rule, if the `deny` flag is set, and an accept rule otherwise.

The links created based on peering policies are of type `IXPLink`. `IXPLink` extends the regular links, tagging them for automatic removal if they are no longer conforming to the policies.

4.2.2 Peering API

The administrator interface generated by Django for our IXP models enables managing IXPs, their member ASes, and configuring peering policies. It is only accessible to SCIONLab administrators though, not to normal users. This is acceptable for managing the IXPs themselves and configuring IXP membership, but AS owners should be able to view and modify their peering policies without going through an administrator. To non-administrator users, the SCIONLab coordinator provides a web interface for creating and deleting user ASes. It would be possible to extend this interface for viewing and editing peering policies, but for our prototype implementation we chose to forgo this option for now. Instead we have extended the coordinator's RESTful API, through which AS configuration updates are deployed.

The unmodified coordinator provides two simple API endpoints at `api/host/<uid>/config` and `api/host/<uid>/deployed_config_version`. The first allows AS hosts, that is hosts running one of the ASes SCION services, to download their configuration via a `GET` request. The host can supply the configuration version it currently runs in the request's query string, so the coordinator only generates a new configuration if the host's version is outdated. Hosts notify the coordinator when a new configuration has been successfully installed trough `POST` requests to the second API endpoint. The `<uid>` in the URLs given above is a random host identifier assigned by the coordinator. Hosts authenticate to the coordinator by providing this identifier and a random secret value. For initial deployment, the API credentials can be retrieved from the coordinator's administrator interface. For user AS operators, the API credentials are included in the initial configuration packet they download during the user AS creation process.

At the moment, there are three ways to install a user AS. It can either run in a pre-configured

VM, be installed from binary packets for Debian based systems, or be installed from source. In the first two cases, the AS's services are managed by systemd. In the last case, the services are run through the process control system supervisord. The configuration API mentioned above does not provide configuration files for the source installation type. Since we are using this installation type in our testbed and want to deploy configurations through the coordinator's API, we added an optional `installation_type` parameter. `installation_type` specifies whether to return a configuration suitable for the packeted version of SCION of for a source installation.

The peering coordinator adds two more API endpoints: `api/host/<uid>/peering_policies` and `api/host/<uid>/peers`. The former enables user AS owners to view, add, and delete peering policies. The later is used to retrieve the list of peering connections resulting from the policies. Authentication is handled by a secret API token in the same way as in the existing APIs. In the following, we provide usage examples of the new API.

Viewing Peering Policies The currently active peering policies of a user AS might be retrieved in the following way:

```
$ curl -X GET -u <uid>:<secret> \
  http://127.0.0.1:8000/api/host/<uid>/peering_policies
{
  "1": {
    "ISD": {
      "accept": [1],
      "deny": [2]
    },
    "AS": {
      "accept": ["ff00:0:212"],
      "deny": ["ff00:0:113"]
    }
  }
}
```

The response is a JSON object containing the peering rules for each IXP the user AS is a member of. In this example, the AS is only connected to one IXP with ID 1. It accepts peering with AS `ff00:0:212` and ISD 1. Peering with AS `ff00:0:113` and ISD 2 is denied. The response can be filtered for a specific IXP by adding `ixp=<id>` to the query string. Note that ASes are identified just by the AS identifier and not by the ISD-AS pair.

For convenience, our testbed framework provides a wrapper around the coordinator's API, simplifying the command to (assuming the host above belongs to AS `1-ff00:0:112`):

```
$ ./ixp-testbed.py policy get 1-ff00:0:112
```

The framework automatically retrieves the API tokens for the specified AS from the coordinator. The query is limited to a single IXP by adding the option `--ixp <id>` to the command. In the following, we present the commands within our evaluation framework only.

Creating Peering Policies The policies retrieved above were create by a `POST` request to the same endpoint:

```
$ ./ixp-testbed.py policy create 1-ff00:0:112 --data \
  '{"1":
     {"AS": {"accept": ["ff00:0:212"], "deny": ["ff00:0:113"]},
      "ISD": {"accept": [1], "deny": [2]}}}'
HTTP/1.1 201 Created
```

The data format is the same as is retrieved via `GET`. Requests to create policies are treated as transactions. If any of the policies is invalid, non of them are created. Creating a policy fails, if the policy exist already or contradicts another as described in Section 3.3.2.

Deleting Peering Policies Policies are deleted via `DELETE` requests:

```
$ ./ixp-testbed.py policy delete 1-ff00:0:112 --data \
  '{"1": {"AS": {"deny": ["ff00:0:113"]}}}'
HTTP/1.1 204 No Content
```

This example deletes the deny AS `ff00:0:113` rule. As with create requests, delete requests are handled as transactions. If deleting any rule fails, the policy set is left unchanged. Note however, that deleting rules is idempotent. A valid rule, that is not currently in the set of policies is deleted successfully.

Getting Information about Peers The list of ASes that accepted peering connections is retrieved with a `GET` request at `api/host/<uid>/peers`. Our framework exposes the API like this:

```
$ ./ixp-testbed.py policy get_peers 1-ff00:0:112
{
  "1": [
    {"label": null, "as_id": "ff00:0:113"},
    {"label": null, "as_id": "ff00:0:212"}
  ]
}
```

In this case, `ff00:0:112` peers at IXP 1 are `ff00:0:113` and `ff00:0:212`. The description of the peer ASes includes the freeform AS label set in the coordinator, which both of the ASes in this example miss. As when retrieving policies, the selection can be limited by supplying `ixp=<id>` in the query string, or the option `--ixp <id>` to our framework.

Chapter 5

Evaluation

In this chapter, we evaluate how well SCION scales with a large number of peering links between user ASes. To this end, we set up a testbed with the framework described in the previous chapter, and measure AS resource consumption as our peering-enabled coordinator adds links.

5.1 Experimental Setup

We generated two topologies for evaluation. The first, which we will call Topology 1 from now on, consists of five ISDs. All ISDs have the same structure. There is one core AS, one attachment point, and 20 user ASes. The core ASes of all five ISDs are connected by fully meshed core links for a total of ten core links in the whole network. The attachment points are connected to their core ASes by a single parent-child link. Attached to the attachment points are the 20 user ASes. Initially, the user ASes have no additional links besides the one connecting them to their attachment point. The structure of an ISD in Topology 1 is visualized in Figure 5.7.

The second topology (Topology 2) also consists of five ISDs with one core AS, one attachment point, and 20 user ASes each. It has the same links as Topology 2, but additionally there are some parent-child links between the user ASes. The structure of the links is identical in every ISD. Figure 5.8 depicts the arrangement. There are five strings of user ASes connected to each other, forming a hierarchy four levels deep. Additionally, there are cross connections between neighboring strings from the second to the third level. Since all user ASes must directly connect to an attachment point, all 20 parent-child links to the attachment point are still there.

Topology 2 might give use some insight into how beaconing behaves when a large number of peering links are added to a more complex topology. After all, intra-ISD beacons are only exchanged on parent-child links, not on peering links. This is also relevant for a future implementation of transitive peering as described in Section 3.2, where parent-child links are introduced in addition to SCION peering links.

Evaluation Procedure To evaluate the scalability of peering links, we add more and more peering links to the user ASes and measure how the CPU usage of the core ASes, the attachment point, and the user ASes changes in reaction. We take 21 CPU usage measurements per AS over the course of the experiment. The first in the initial configuration with no peering links. After that, we create more peering links by installing an open peering policy in five user ASes, one in every ISD. By open peering policy we mean accepting peering with all other user ASes. Thus, the five ASes all peer with each other in a fully meshed network of peering links. We then take another CPU usage measurement. This procedure is repeated 19 more times. In every pass, we add five more ASes, one from every ISD, to the set of open peers and measure the CPU usage with the now increased number of peering links.

The order we add user ASes in the same ISD to the open peering set is the same as given by the alphabetic order of user ASes in Figure 5.7 and Figure 5.8. CPU usage measurements are taken as average over a time span of 5 minutes immediately after the updated AS configurations have been installed. The intra-ISD beaconing interval is set to 5 seconds, thus we capture about

60 beaconing intervals in one measurement. Inter-ISD beacons are propagated at a slower rate of once in every 60 seconds, therefore we capture 5 beaconing intervals in 5 minutes. 60 s inter-ISD and 5 s intra-ISD beaconing intervals are the default values of the SCIONLab coordinator.

Since every user AS joining the already peering ones immediately connects to every other AS already peering, the total number of peering links rises quadratically in every step of the experiment. More precisely, during the n-th CPU usage measurement there are $\frac{25n^2-5n}{2}$ peering links in the entire network. The number of peering links per ISD, by which we mean links with at least one endpoint in the same ISD, is $\frac{9n^2-n}{2}$. This is the amount of peering links the intra-ISD beaconing has to deal with in every ISD. The number of peering links the first user ASes in every ISD (`A` in Figure 5.7 and Figure 5.8) has is $\max(0, 5n-1)$ during the n-th measurement.

Our evaluation only involves control plane traffic between the beaconing and paths servers. There are no packets exchanged between SCION end hosts.

Hardware We ran the evaluation on a single machine equipped with an 18-core/36-thread Intel Xeon Gold 6150 processor and 512 GiB of main memory. The total peak CPU utilization we observed was below 50% of all processor threads, indicating enough total processing power was available to not impact the results. The combined memory consumption of all ASes peaked at around 32 GiB, so the amount of memory was not a bottleneck either.

5.2 Results

Figure 5.1 shows the CPU utilization of one of the core ASes in Topology 1 plotted over the number of peering links, which connect to at least one AS in the ISD of the core AS. CPU usage is given for the whole AS container (cntr. total), the beacon server alone, the path server alone, and the border routers alone. The CPU usage is given in percent normalized to a single logical processor core, i.e., the maximum CPU usage on our 36-thread evaluation system would be 3600%. Total container CPU usage includes the beacon server, the path server, the border routers, and other processes necessary to run SCION, like the certificate server.

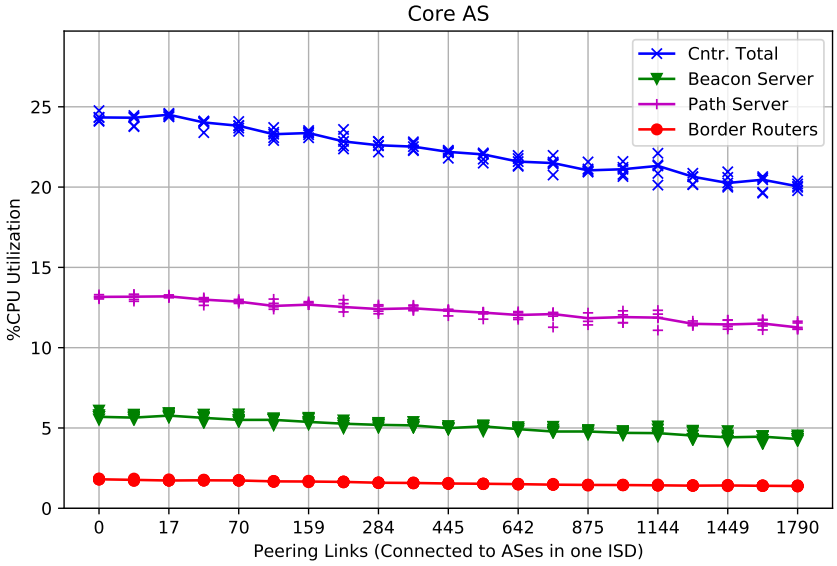


Figure 5.1: CPU usage of a core AS in Topology 1.

The graph shows every data point we acquired for the AS during five repetitions of the evaluation process described above. The x-axis is evenly spaced with respect to the measurement number and labeled with the resulting link count in one ISD. The lines we plotted connect the medians of the five measurements taken per link count.

As we can see, the CPU usage of the path and beacon server decrease slightly with a rising number of peering links. We would have expected the CPU usage to stay stable or rise slightly, because the number of beacons and path segments registered should stay constant. Peering links are only added as additional information to beacons, which would be propagated anyway. Likewise, they do not create their own path segments, but only increase the size of the existing segments.

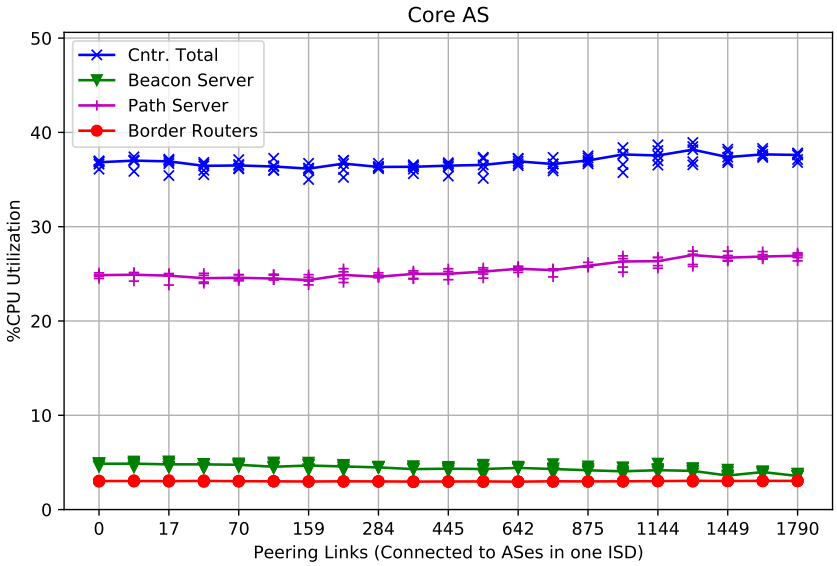


Figure 5.2: CPU usage of a core AS in Topology 2.

If we compare Figure 5.1 with Figure 5.2, depicting the CPU usage of a core AS in Topology 2, we see that the CPU usage of the beacon server is on a similar absolute level of around 5% and also slightly decreases. However, the CPU usage of the path server is slightly increasing. The higher level of the path server's CPU usage is expected, because there are much more path segments being registered in Topology 2.

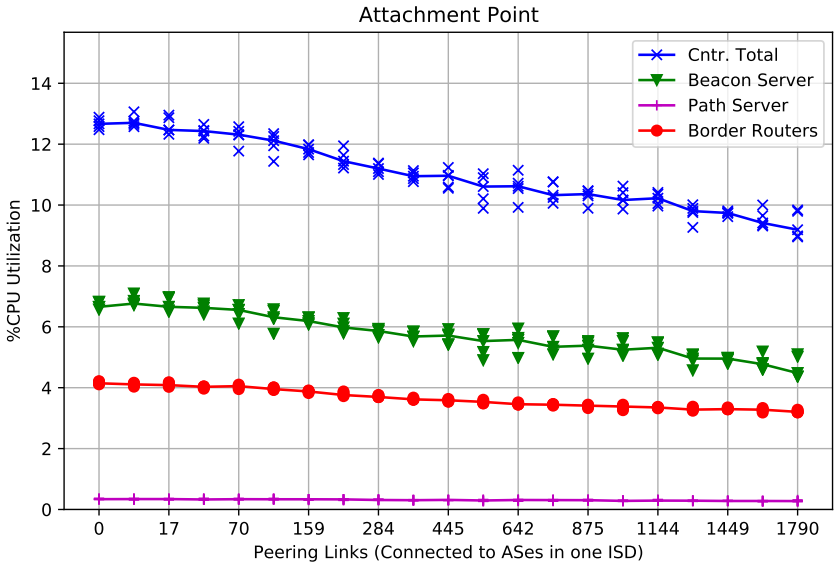


Figure 5.3: CPU usage of an attachment point AS in Topology 1.

Looking at the statistics of the attachment points in Figure 5.3 and Figure 5.4, reveals a downward trend in CPU usage of the beacon server in both topologies. Note that the path server

in the attachment points is much less active than in the core ASes. The core path server has a high CPU load, because non-core ASes register down path-segment at this path server, whereas path servers in non-core ASes only have to deal with local up-segments.

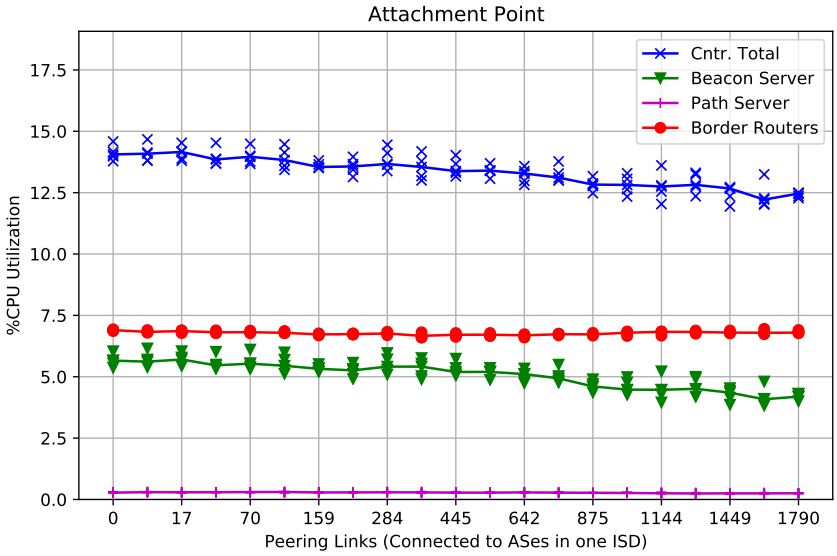


Figure 5.4: CPU usage of an attachment point AS in Topology 2.

Finally, Figure 5.5 and Figure 5.6 depict the CPU utilization of a user AS. The AS in question is labeled `A` in Figure 5.7 and Figure 5.8, respectively.

The graphs are almost identical, but there is one difference. The CPU load of the beacon server and the border routers is slightly higher in Topology 2 than in Topology 1. This can most easily be seen for the initial state with no peering links, were the beacon server's CPU usage is at 2% in Topology 1, and at around 2.5% in Topology 2. The higher load in Topology 2 is due the the child AS labeled `F` in Figure 5.8, to which beacons have to be propagated.

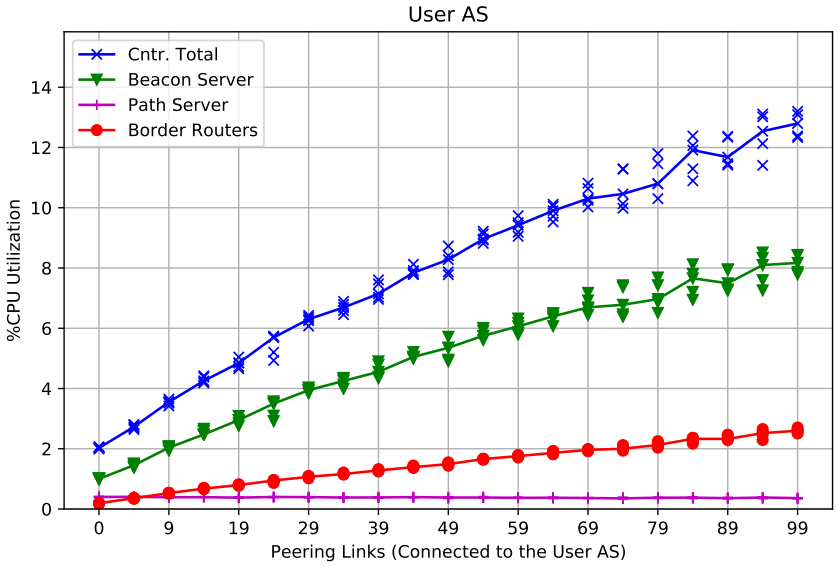


Figure 5.5: CPU usage of a user AS in Topology 1. The depicted AS is one of the first five to begin peering, i.e., at position `A` in Figure 5.7.

In both topologies, the CPU load of the path server increases linearly with the amount of

peering links the user AS has. The load of the border routers rises accordingly, indicating the now increased control plane traffic. As in the attachment points, the path servers have little to do without any actual end hosts retrieving path segments and idle at a low CPU utilization.

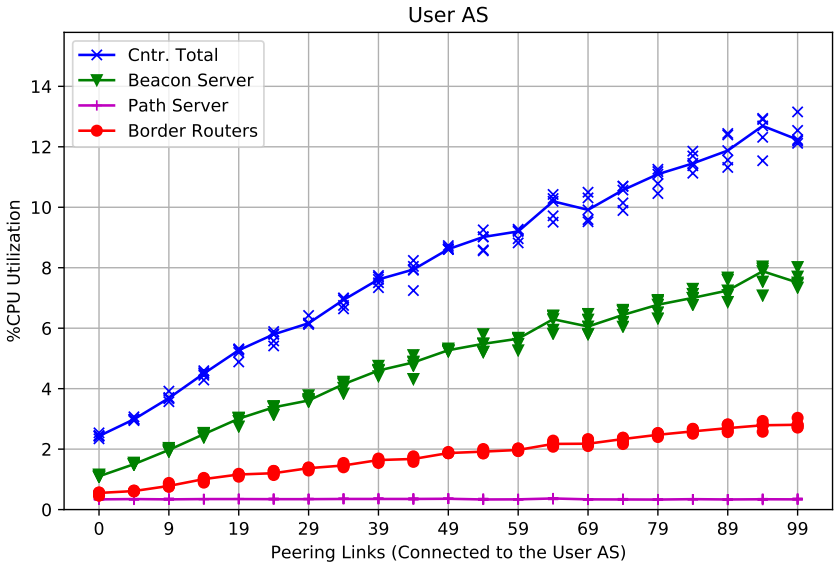


Figure 5.6: CPU usage of a user AS in Topology 2. The depicted AS is one of the first five to begin peering, i.e., at position `A` in Figure 5.8.

The memory consumption of core ASes in Topology 1 peaked at around 300 MiB, the attachment points allocated up to 350 MB, and the user AS 200 MB. In Topology 2, the maximum memory consumption of a core AS was around 350 MiB. Attachment points needed up to 450 MiB, and user ASes peaked at 300 MiB.

Figure 5.7 and Figure 5.8 visualized the relative increase in CPU utilization from the initial state of no user AS peering to the final state of fully meshed peering links. Depicted are the core AS, the attachment point (AP), and user ASes (`A` to `T`) of ISD 1. As we have already assessed, the CPU usage of the core and AP AS in Topology 1 drops with more peering links. Quantitatively, the reduction in CPU load is 17.5% in the core AS, and 26.2% in the attachment point. Since the topology is symmetric with respect to the user ASes, they all show the same rise in CPU utilization of about 84%.

In Topology 2, the CPU load of the core AS increases by 2%. The load of the attachment point decreases by 12%. Since the user ASes in Topology 2 have different numbers of parent and child ASes, the increase in CPU usage differs. Interestingly, the highest increase in CPU usage was observed in ASes `A` to `E`. The increase by around 81% is similar to the values from the user ASes of Topology 1. These ASes are unified by the fact, that they all have only a single parent AS, the attachment point. The slightly lower load increase in Topology 2 probably stems from a higher base load due to the additional links to child ASes `F` to `J`.

The lowest load increase happens in ASes `K` to `O`, which have 3 to 4 parent ASes and one child each. These ASes have the highest CPU consumption of 7.5 to 9.5% without peering links. Therefore, it appears the CPU utilization increases less, if the initial beaconing overhead was higher. In absolute numbers, however, ASes `K` to `O` have the highest total CPU load of around 20% after all peering links have been added. This is consistent with the fact, that they have to process the largest amount of beacons.

5.3 Discussion and Conclusions

The evaluation in this chapter only provides a general indication of the performance to be expected from the current SCION implementation in an IXP setting. Most importantly, we have only

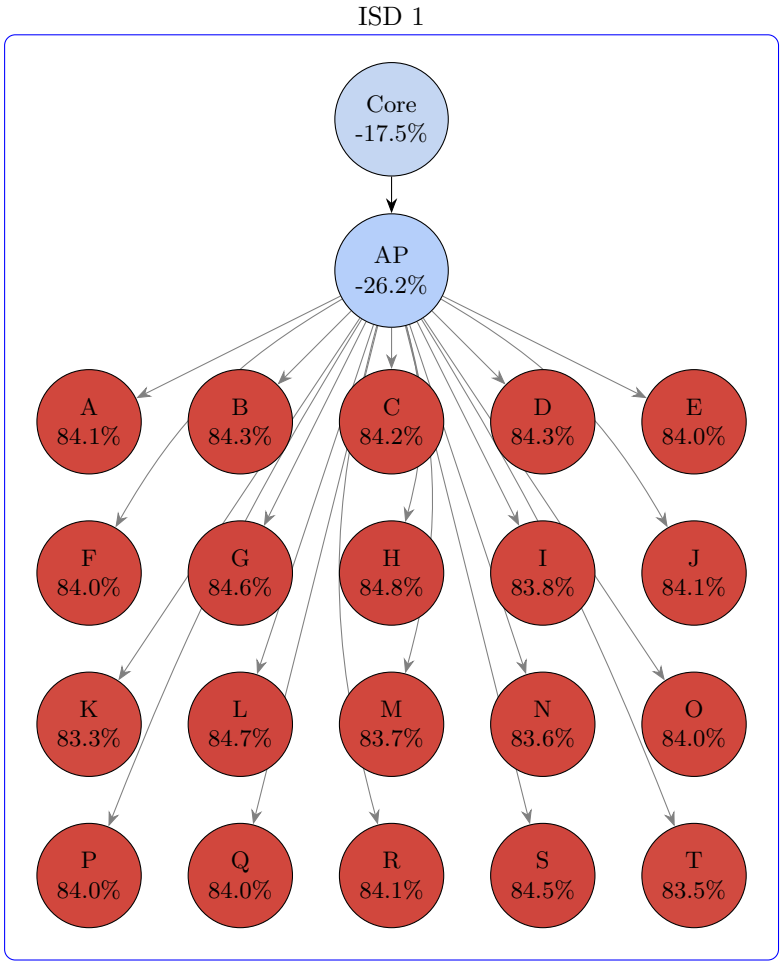


Figure 5.7: Relative increase in CPU usage from no peering links to fully meshed peering links in Topology 1.

generated control plane traffic with the beaconing, there was no actual end host data exchanged. Actually transmitting data would certainly have an impact on the resource consumption of the path servers, because they would have to supply path segments to end hosts.

The curious drop in CPU usage of the core AS and attachment point with a higher number of peering links still has to be investigated. The beacon server exposes statistics that could reveal whether the amount of beacons processed drops, or the amount of work done per beacon changes. Similarly, the load differences in the user ASes of Topology 2 should be analyzed in detail. It would be useful to know how well peering links scale for ASes with certain amounts of parent and child links. For example, how does the load of the beacon server increase when many beacons containing peering links arrive, and how does it increase, if many peering links are added to the beacons send to child ASes.

Nevertheless, our preliminary results indicate, that SCION scales well with the peering method we implemented in the SCIONLab coordinator. The core and attachment point ASes are not overwhelmed by a quadratically growing number of peering links per ISD, and the user ASes scale linearly with the amount of peers they have.

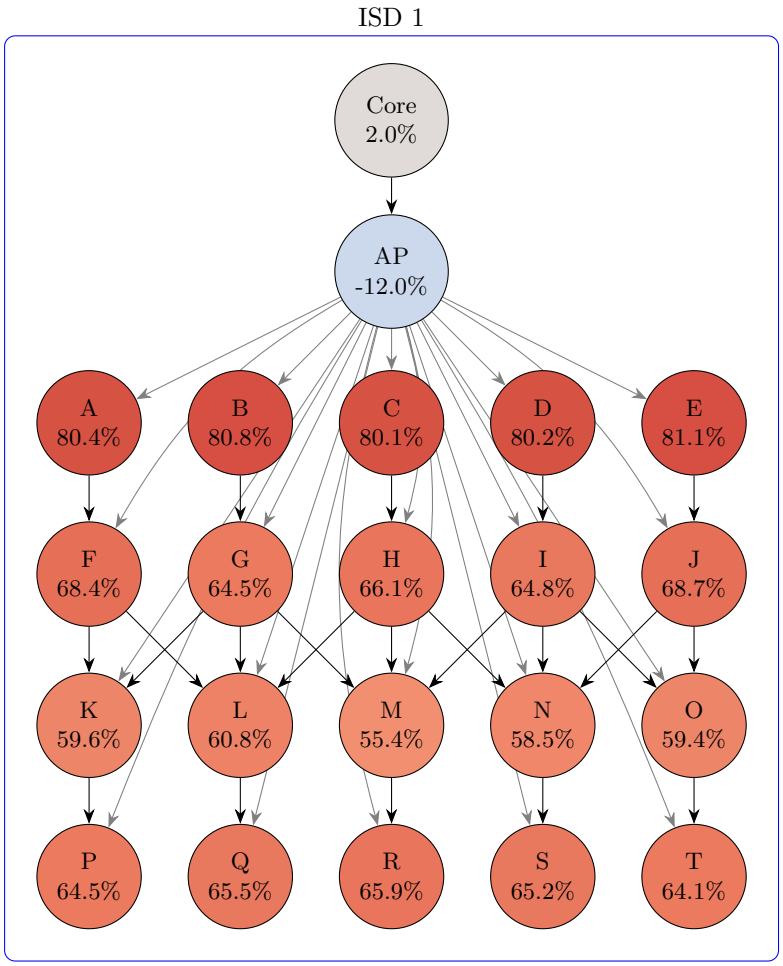


Figure 5.8: Relative increase in CPU usage from no peering links to fully meshed peering links in Topology 2.

Chapter 6

Future Work

This work has touch a broad range of topics, that could be developed further. In Chapter 2, we described SCION's current support for path policies in form of beacon and path segment filters. Deploying SCION at an IXP would likely require more sophisticated policies, ideally enforceable in the data plane. It should be straight forward to extend the current beacon filters to be specific to certain child/customer ASes, as well as to include peering links only selectively. Beaconing extensions communicating policies directly could provide even more flexibility in the future. It will be important to keep the requirements of AS operator in mind, when designing these extensions.

In Chapter 3 we identified transitive peering as a use case not well supported by SCION and sketched a workaround involving parent-child links as replacement for peering links. In the future, a SCION peering coordination service should be able to create such links. This will require developing an algorithm transforming a given policy specification to a suitable SCION AS graph. In tandem with more sophisticated policy mechanism in the beaconing process, this should cover a wide range of IXP use cases.

The modifications we made to the SCIONLab coordinator cover only simple peering rules and only apply to user ASes, which typically are leaf ASes. It would be useful to extend the coordinator's peering concepts to more well-connected ASes, including core ASes. Once the more refined beaconing and path registration filters mentioned above are available, the coordinator could incorporate these in the AS configurations it generates. Combined with a way to express more complex peering intentions than the accept and deny rules we have now, this could open up most peering scenarios covered by BGP today in addition to SCION specific features like hidden paths.

At the moment, there is only a single instance of the coordinator in the SCIONLab network. When SCION is deployed at an IXP, IXP operators might want to run their own peering coordinator. This coordinator must be able to cooperate with other coordinators in the network, requiring a redesign of the current implementation. Ultimately, ASes should manage their own configuration files instead of downloading them from a coordinator, which should only have an advisory function, like a BGP route server.

Our evaluation of SCION peering at an IXP is still incomplete. In the future, we would like to extend our testbed to a more realistic network hosted on more than a single server. This would enable generating actual traffic between end hosts to examine the scalability of SCION's path construction process more closely.

Another topic, we have touched on, is integrating SCION in the data plane of an IXP. With the new SCION header discussed in Section 3.4, bare-metal switches might become a corner stone of this integration.

Chapter 7

Conclusion

From our analysis of peering in SCION, we conclude, that more work in the area of SCION path policies is needed to fully support all peering relations possible at IXPs. Nevertheless, SCION can already support many typical peering business models. We also identified the possibility of introducing SCION into an IXP's data plane by utilizing the custom header field support of some bare-metal switches.

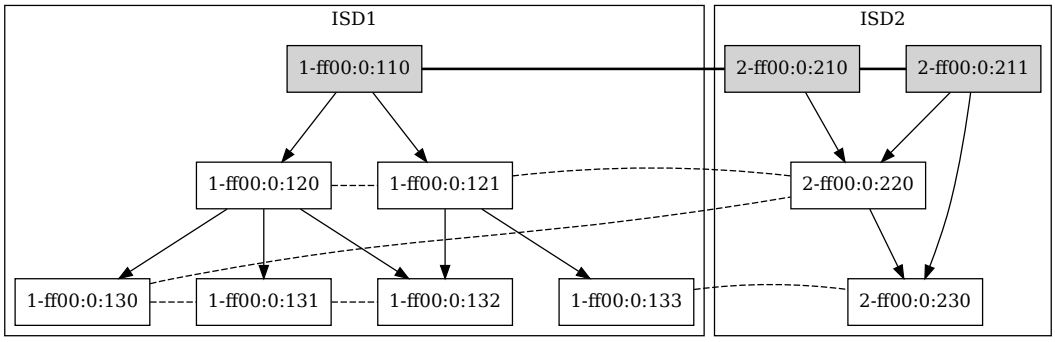
Our prototype peering coordinator demonstrates, how an open SCION peering ecosystem could look like by providing automatic peering links between user ASes based on simple accept and deny policy rules. Even though our evaluation is limited in scope, it seems that SCION's beaconing process scales well with the many peering links created by our coordinator.

Appendix A

Example Topologies

A.1 Complex Topology

Topology used as running example in Section 3.1.



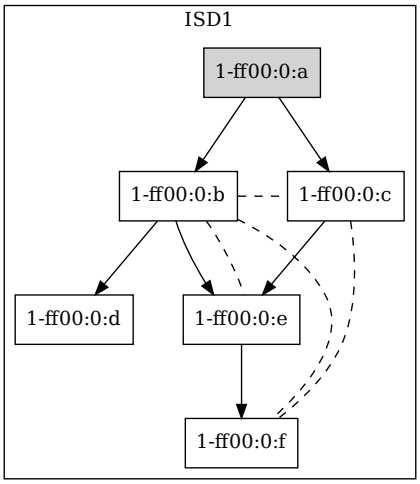
Listing A.1: Topology definition.

```
defaults:
  subnet: "127.0.0.0/8"
  link_subnet: "10.1.10.0/24"
  zookeepers: {1: {addr: 127.0.0.1}}
networks:
  "links":
    type: "docker_bridge"
    subnet: "10.1.20.0/24"
ASes:
  "1-ff00:0:110": {core: true}                    # A
  "1-ff00:0:120": {cert_issuer: "1-ff00:0:110"} # B
  "1-ff00:0:121": {cert_issuer: "1-ff00:0:110"} # C
  "1-ff00:0:130": {cert_issuer: "1-ff00:0:110"} # D
  "1-ff00:0:131": {cert_issuer: "1-ff00:0:110"} # E
  "1-ff00:0:132": {cert_issuer: "1-ff00:0:110"} # F
  "1-ff00:0:133": {cert_issuer: "1-ff00:0:110"} # G
  "2-ff00:0:210": {core: true}                    # H
  "2-ff00:0:211": {core: true}                    # I
  "2-ff00:0:220": {cert_issuer: "2-ff00:0:210"} # J
  "2-ff00:0:230": {cert_issuer: "2-ff00:0:210"} # K
links:
  # Core
  - {a: "1-ff00:0:110", b: "2-ff00:0:210", linkAtoB: CORE, network: "links"}
  - {a: "2-ff00:0:210", b: "2-ff00:0:211", linkAtoB: CORE, network: "links"}
  # ISD 1 Parent-Child
  - {a: "1-ff00:0:110", b: "1-ff00:0:120", linkAtoB: CHILD, network: "links"}
  - {a: "1-ff00:0:110", b: "1-ff00:0:121", linkAtoB: CHILD, network: "links"}
  - {a: "1-ff00:0:120", b: "1-ff00:0:130", linkAtoB: CHILD, network: "links"}
  - {a: "1-ff00:0:120", b: "1-ff00:0:131", linkAtoB: CHILD, network: "links"}
  - {a: "1-ff00:0:120", b: "1-ff00:0:132", linkAtoB: CHILD, network: "links"}
```

```
31    - {a: "1-ff00:0:121", b: "1-ff00:0:132", linkAtoB: CHILD, network: "links"}
32    - {a: "1-ff00:0:121", b: "1-ff00:0:133", linkAtoB: CHILD, network: "links"}
33    # ISD 2 Parent-Child
34    - {a: "2-ff00:0:210", b: "2-ff00:0:220", linkAtoB: CHILD, network: "links"}
35    - {a: "2-ff00:0:211", b: "2-ff00:0:220", linkAtoB: CHILD, network: "links"}
36    - {a: "2-ff00:0:211", b: "2-ff00:0:230", linkAtoB: CHILD, network: "links"}
37    - {a: "2-ff00:0:220", b: "2-ff00:0:230", linkAtoB: CHILD, network: "links"}
38    # Peering
39    - {a: "1-ff00:0:120", b: "1-ff00:0:121", linkAtoB: PEER, network: "links"}
40    - {a: "1-ff00:0:121", b: "2-ff00:0:220", linkAtoB: PEER, network: "links"}
41    - {a: "1-ff00:0:130", b: "1-ff00:0:131", linkAtoB: PEER, network: "links"}
42    - {a: "1-ff00:0:131", b: "1-ff00:0:132", linkAtoB: PEER, network: "links"}
43    - {a: "1-ff00:0:133", b: "2-ff00:0:230", linkAtoB: PEER, network: "links"}
44    - {a: "1-ff00:0:130", b: "2-ff00:0:220", linkAtoB: PEER, network: "links"}
45  CAs:
46    CA1-1: {ISD: 1, commonName: CA1-1}
47    CA2-1: {ISD: 2, commonName: CA2-1}
```

A.2 Peering With Customers

Example topology from Section 3.1.6.



Listing A.2: Topology definition.

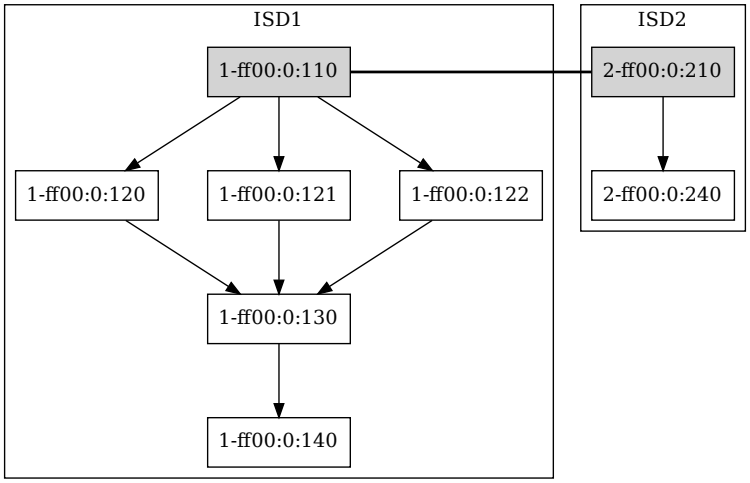
```
1   defaults:
2     subnet: "127.0.0.0/8"
3     link_subnet: "10.1.10.0/24"
4     zookeepers: {1: {addr: 127.0.0.1}}
5   ASes:
6     "1-ff00:0:A": {core: true}
7     "1-ff00:0:B": {cert_issuer: "1-ff00:0:A"}
8     "1-ff00:0:C": {cert_issuer: "1-ff00:0:A"}
9     "1-ff00:0:D": {cert_issuer: "1-ff00:0:A"}
10    "1-ff00:0:E": {cert_issuer: "1-ff00:0:A"}
11    "1-ff00:0:F": {cert_issuer: "1-ff00:0:A"}
12  IXPs:
13    "ixp1": {network: "10.1.20.0/24"}
14  links:
15    - {a: "1-ff00:0:A", b: "1-ff00:0:B", linkAtoB: CHILD}
16    - {a: "1-ff00:0:A", b: "1-ff00:0:C", linkAtoB: CHILD}
17    - {a: "1-ff00:0:B", b: "1-ff00:0:D", linkAtoB: CHILD}
18    - {a: "1-ff00:0:B", b: "1-ff00:0:E", linkAtoB: CHILD}
19    - {a: "1-ff00:0:C", b: "1-ff00:0:E", linkAtoB: CHILD}
20    - {a: "1-ff00:0:E", b: "1-ff00:0:F", linkAtoB: CHILD}
```

```
21    - {a: "1-ff00:0:B", b: "1-ff00:0:C", linkAtoB: PEER, network: "ixp1"}
22    - {a: "1-ff00:0:B", b: "1-ff00:0:E", linkAtoB: PEER, network: "ixp1"}
23    - {a: "1-ff00:0:B", b: "1-ff00:0:F", linkAtoB: PEER, network: "ixp1"}
24    - {a: "1-ff00:0:C", b: "1-ff00:0:F", linkAtoB: PEER, network: "ixp1"}
25  CAs:
26    CA1-1: {ISD: 1, commonName: CA1-1}
```

A.3 Beacon Server Policy

Topology definition and policy files for the example in Section 2.4.

To apply the policies, append the policy configuration in Listing A.4 to the configuration file of AS `1-ff00:0:130`'s beacon server. The configuration file is located in the testbed working directory under `1-ff00_0_130/gen/ISD1/ASff00_0_130/bs1-ff00_0_130-1/bs.toml`. Then create files `propagation_policy.yaml`, `up_policy.yaml`, and `down_policy.yaml` in the same directory, containing Listing A.5, Listing A.6, and Listing A.7, respectively.



Listing A.3: Topology definition.

```
1   defaults:
2     subnet: "127.0.0.0/8"
3     link_subnet: "10.1.10.0/24"
4     zookeepers: {1: {addr: 127.0.0.1}}
5   ASes:
6     "1-ff00:0:110": {core: true}                    # A
7     "1-ff00:0:120": {cert_issuer: "1-ff00:0:110"} # B
8     "1-ff00:0:121": {cert_issuer: "1-ff00:0:110"} # C
9     "1-ff00:0:122": {cert_issuer: "1-ff00:0:110"} # D
10    "1-ff00:0:130": {cert_issuer: "1-ff00:0:110"} # E
11    "1-ff00:0:140": {cert_issuer: "1-ff00:0:110"} # F
12    "2-ff00:0:210": {core: true}                    # G
13    "2-ff00:0:240": {cert_issuer: "2-ff00:0:210"} # H
14  links:
15    - {a: "1-ff00:0:110", b: "2-ff00:0:210", linkAtoB: CORE}
16    - {a: "1-ff00:0:110", b: "1-ff00:0:120", linkAtoB: CHILD}
17    - {a: "1-ff00:0:110", b: "1-ff00:0:121", linkAtoB: CHILD}
18    - {a: "1-ff00:0:110", b: "1-ff00:0:122", linkAtoB: CHILD}
19    - {a: "1-ff00:0:120", b: "1-ff00:0:130", linkAtoB: CHILD}
20    - {a: "1-ff00:0:121", b: "1-ff00:0:130", linkAtoB: CHILD}
21    - {a: "1-ff00:0:122", b: "1-ff00:0:130", linkAtoB: CHILD}
22    - {a: "1-ff00:0:130", b: "1-ff00:0:140", linkAtoB: CHILD}
23    - {a: "2-ff00:0:210", b: "2-ff00:0:240", linkAtoB: CHILD}
```

```
24 CAs:
25   CA1-1: {ISD: 1, commonName: CA1-1}
26   CA2-1: {ISD: 2, commonName: CA2-1}
```

Listing A.4: Beacon server configuration for AS `1-ff00:0:130`.

```
1 [bs.policies]
2 Propagation = "gen/ISD1/ASff00_0_130/bs1-ff00_0_130-1/propagation_policy.yaml"
3 UpRegistration = "gen/ISD1/ASff00_0_130/bs1-ff00_0_130-1/up_policy.yaml"
4 DownRegistration = "gen/ISD1/ASff00_0_130/bs1-ff00_0_130-1/down_policy.yaml"
```

Listing A.5: Beacon propagation policy.

```
1 BestSetSize: 5
2 CandidateSetSize: 100
3 MaxExpTime: 63
4 Filter:
5   MaxHopsLength: 10
6   AsBlackList: ["ff00:0:120"]
7   IsdBlackList: []
8   AllowIsdLoop: false
9 Type: "Propagation"
```

Listing A.6: Up-segment registration policy.

```
1 BestSetSize: 5
2 CandidateSetSize: 100
3 MaxExpTime: 63
4 Filter:
5   MaxHopsLength: 10
6   AsBlackList: ["ff00:0:121"]
7   IsdBlackList: []
8   AllowIsdLoop: false
9 Type: "UpSegmentRegistration"
```

Listing A.7: Down-segment registration policy.

```
1 BestSetSize: 5
2 CandidateSetSize: 100
3 MaxExpTime: 63
4 Filter:
5   MaxHopsLength: 10
6   AsBlackList: ["ff00:0:122"]
7   IsdBlackList: []
8   AllowIsdLoop: false
9 Type: "DownSegmentRegistration"
```

Bibliography

[1] Ruwaifa Anwar, Haseeb Niaz, David Choffnes, Ítalo Cunha, Phillipa Gill, and Ethan Katz-Bassett. Investigating interdomain routing policies in the wild. In *Proceedings of the Internet Measurement Conference*, pages 71–77, 2015.

[2] David Barrera, Laurent Chuat, Adrian Perrig, Raphael M. Reischuk, and Pawel Szalachowski. The SCION internet architecture. In *Communications of the ACM*, volume 60, pages 56–65, 2017.

[3] Kevin Butler, Toni R. Farley, Patrick McDaniel, and Jennifer Rexford. A survey of BGP security issues and solutions. *Proceedings of the IEEE*, 98(1):100–122, 2010.

[4] R. Chandra, P. Traina, Cisco Systems, and T. Li. BGP communities attribute. IETF RFC 1997, 1996.

[5] DE-CIX Operational BGP Communities. `https://www.de-cix.net/de/resources/route-server-guides/operational-bgp-communities`. Visited on March 16th, 2020.

[6] Christoph Dietzel, Anja Feldmann, and Thomas King. Blackholing at IXPs: On the effectiveness of DDoS mitigation in the wild. In *International Conference on Passive and Active Measurement*, pages 319–332. Springer International Publishing, 2016.

[7] Christoph Dietzel, Matthias Wichtlhuber, Georgios Smaragdakis, and Anja Feldmann. Stellar: Network attack mitigation using advanced blackholing. In *Proceedings of the International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '18, pages 152–164. ACM, 2018.

[8] Django web framework. `https://www.djangoproject.com/`.

[9] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, 2001.

[10] Phillipa Gill, Michael Schapira, and Sharon Goldberg. A survey of interdomain routing policies. *ACM SIGCOMM Computer Communication Review*, 44(1):28–34, 2013.

[11] Vasileios Giotsas, Matthew Luckie, Bradley Huffaker, and kc claffy. Inferring complex AS relationships. In *Proceedings of the Internet Measurement Conference*, pages 23–30, 2014.

[12] Sharon Goldberg. Why is it taking so long to secure internet routing? *Communications of the ACM*, 57(10):56–63, 2014.

[13] J. Heitz, Cisco, J. Snijders, NTT, K. Patel, Arrcus, I. Bagdonas, Equinix, N. Hilliard, and INEX. BGP large communities attribute. IETF RFC 8092, 2017.

[14] Internet routing registry. `http://www.irr.net/`.

[15] ISD and AS numbering. `https://github.com/scionproto/scion/wiki/ISD-and-AS-numbering`. Visited on March 16th, 2020.

[16] M. Lepinski and S. Kent. An infrastructure to support secure internet routing. IETF RFC 6480, 2012.

[17] Aemen Lodhi, Natalie Larson, Amogh Dhamdhere, Constantine Dovrolis, and kc claffy. Using PeeringDB to understand the peering ecosystem. *ACM SIGCOMM Computer Communication Review*, 44(2):20–27, 2014.

[18] P. Mohapatra, J. Scudder, D. Ward, R. Bush, and R. Austein. BGP trefix origin validation. IETF RFC 6811, 2013.

[19] William B. Norton. *The Internet Peering Playbook*. DrPeering Press, 2014 edition, 2014.

[20] PeeringDB. `https://www.peeringdb.com/`.

[21] Adrian Perrig, Pawel Szalachowski, Raphael M. Reischuk, and Laurent Chuat. *SCION: A Secure Internet Architecture*. Springer International Publishing AG, 2017.

[22] Y. Rekhter, T. Li, and S. Hares. A border gateway protocol 4 (BGP-4). IETF RFC 4271, 2006.

[23] Philipp Richter, Georgios Smaragdakis, Anja Feldmann, Nikolaos Chatzis, Jan Boettger, and Walter Willinger. Peering at Peerings: On the Role of IXP Route Servers. In *Proceedings of the Internet Measurement Conference*, pages 31–44. ACM, 2014.

[24] S. Sangli, D. Tappan, Cisco Systems, Y. Rekhter, and Juniper Networks. BGP extended communities attribute. IETF RFC 4360, 2006.

[25] SCIONLab. `https://www.scionlab.org/`.

[26] SCIONLab user interface and administration. `https://github.com/netsec-ethz/scionlab`.

[27] Vadim Tkachenko. Testing docker multi-host network performance. `https://www.percona.com/blog/2016/08/03/testing-docker-multi-host-network-performance/`, 2016. Visited on March 16th, 2020.

STATEMENT OF AUTHORSHIP

Thesis: Deployment of SCION on Internet Exchange Point Infrastructure

Name:	Lars-Christian	Surname:	Schulz
Date of birth:	23.08.1993	Matriculation no.:	200061

I herewith assure that I wrote the present thesis independently, that the thesis has not been partially or fully submitted as graded academic work and that I have used no other means than the ones indicated. I have indicated all parts of the work in which sources are used according to their wording or to their meaning.

I am aware of the fact that violations of copyright can lead to injunctive relief and claims for damages of the author as well as a penalty by the law enforcement agency.

Magdeburg, 17.03.2020

_____