# 12 Host Structure

SERGIU COSTEA, MATTHIAS FREI, MARTEN GARTNER, DAVID HAUSHEER, THORBEN KRÜGER, JORDI SUBIRA NIETO, FRANÇOIS WIRZ*

## Chapter Contents



In this chapter, we discuss how SCION-enabled end hosts can benefit from SCION properties. For native communication over SCION to be possible, hosts are expected to have a few software components installed and configured. Nevertheless, as we show in Chapter 13, there are mechanisms to ensure that even legacy (i.e., non-updated) hosts can benefit from SCION.

We first introduce how SCION-enabled hosts can communicate over SCION today (§12.1), then we provide an outlook on how we envision SCION-based communication in the future (§12.2).

## 12.1 Host Components

We introduce the main host software components that enable applications to natively communicate via SCION.

*This chapter reuses content from Lee, Perrig, and Szalachowski [322].

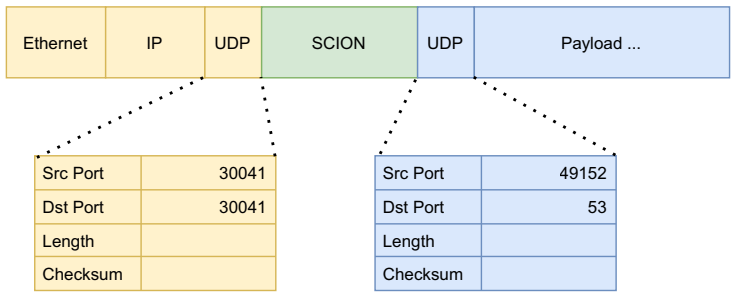


Figure 12.1: The packet layout of a SCION packet containing a UDP payload, transported over UDP/IP/Ethernet. This is how SCION packets are communicated within an AS. The UDP/IP underlay (yellow) shows the fixed port 30041.

The central component of the host structure is the dispatcher (§12.1.1), which handles all incoming and outgoing SCION packets. The SCION daemon (§12.1.2) handles control-plane messages (e.g., to fetch paths to remote ASes). Furthermore, to encourage developers to write SCION applications, the snet library (§12.1.3) provides a simple interface for sending and receiving SCION packets. Binary packages are available[1] for the main components, so that SCION can be easily installed with a simple package manager.

We then present the SCION app for Android (§12.1.4), which demonstrates the portability of software components thanks to their implementation in Go.

### 12.1.1 SCION Dispatcher

In the current implementation of SCION, we use a UDP/IP underlay to communicate between nodes within an AS: All SCION packets, from end hosts to routers, routers to end hosts, and between two end hosts of the same AS, are communicated through a designated UDP port (30041). Figure 12.1 shows the layout of a SCION packet communicated via UDP/IP as the underlying network.

The central component of the SCION host stack is the Dispatcher, a single process within each host that handles all SCION packets on this designated UDP port. Its main role is to receive incoming packets and deliver them to individual SCION applications. This setup is shown in Figure 12.2.

When applications on the local system want to send or receive traffic on a UDP/SCION address, they register the desired local UDP/SCION address with the dispatcher. The dispatcher maintains a table of all registrations, which it uses to look up where to forward traffic whose destination contains a local

[1] The SCION Debian packages are available here: `https://packages.netsec.inf.ethz.ch/debian/`

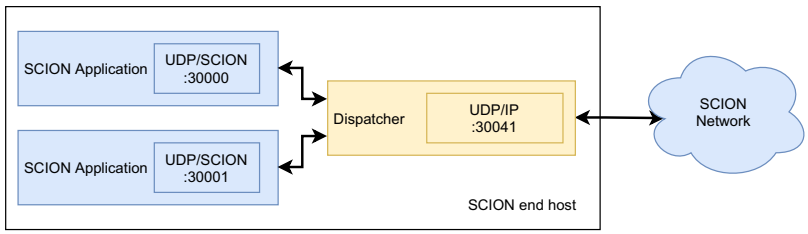


Figure 12.2: SCION Dispatcher overview.

address. Both the registration and future data traffic between the application and the dispatcher go through a UNIX domain socket. Once registered with the dispatcher, applications can start sending and receiving UDP/SCION traffic.

When sending packets, the dispatcher will take care of encapsulating the SCION packet with a UDP/IP header and then send it out on the network towards the destination. The destination is either a SCION Router or a SCION-enabled host in the local AS.

When receiving packets, the dispatcher consults the registration address table to determine to which application the packet is destined, and then delivers the packet to the application. For example, in the case shown in Figure 12.2, if a SCION packet with destination port 30000 arrives, it is sent to the application represented by the upper box in the figure.

When an application closes one of its connections, it closes the UNIX domain socket; the dispatcher removes the associated registration from its tables.

In addition to forwarding, the dispatcher handles some SCMP-specific tasks:

- If an SCMP packet received from the network is for a local application (e.g., a ping reply), it will select the entry in the registration table based on information in the SCMP packet and deliver it. On the application side, snet (§12.1.3) takes care of the packet depending on application policy.
- If an SCMP packet received from the network is for the local host (e.g., a remote system is sending a ping to the local system), then the dispatcher processes the packet itself and, if needed, replies to the sender.

The Dispatcher code is executed in userspace. We envision that in the future, the functionality of the dispatcher will be executed inside a kernel module or as an eBPF program, as discussed in §12.2.6.

### 12.1.2 SCION Daemon

The SCION daemon is a background process running on end hosts with the goals of (1) handling SCION control-plane messages, and (2) providing an

API for applications and libraries to interact with the SCION control plane. Specifically, the SCION daemon implements the following services:

- **Path lookup:** Provides path lookup functionality for host applications. The path lookup process is described in §4.5, and the path segment combination process is described in §5.5. During this process, the SCION daemon validates the individual path segments based on the control-plane PKI. The path segments are cached here, so that information can be shared across all SCION applications running on a host.
- **Topology information:** Provides information about the topology of the local AS. Topology information includes addresses of border routers (with their interface identifiers) and information on running services (e.g., RHINE (§19.3) or path servers).
- **Extensions:** Various SCION extensions and sub-protocols, such as COLIBRI (§10.2) and EPIC (§10.1), implement their control plane as part of the SCION daemon and extend its API.

### 12.1.3 Application Networking Library (snet)

Snet is a Go library[2] that hides away the complexity of the underlying SCION stack behind a simple interface for sending and receiving network traffic. The library achieves this by implementing the Go standard library interfaces for network connections: `net.PacketConn` and `net.Conn`. With snet, developers can write SCION network applications quickly, as they only need to focus on the high-level functionality of their application. The compatibility with Go standard library interfaces also enables developers to use SCION network connections in many third-party frameworks such as servers or RPC frameworks. It is also possible to reuse existing networking libraries for SCION. For example, the squic sub-package of snet provides an implementation of QUIC over UDP/SCION by integrating the quic-go library [353].

Currently, the library includes full support for UDP/SCION communication and several utility functions for interacting with SCMP and the SCION daemon. Critical to UDP/SCION support is the SCION dispatcher, the local process that handles the forwarding of data between SCION hosts, described in §12.1.1.

We illustrate snet operations by taking a look at the lifetime of a snet UDP/SCION connection. When an application creates a new UDP/SCION connection object, snet contacts the SCION dispatcher on the local system and informs it of the application's desired local address-port pair. The dispatcher registers this pair, and will later use it to deliver SCION packets.

[2]The documentation for snet can be found at `https://pkg.go.dev/github.com/scionproto/scion/go/lib/snet`

If the registration is successful, snet gives the application a handle to the new connection. The application can now use this handle to send or receive packets. Whenever snet needs to send data on the SCION network, it serializes the payload and prepends the necessary headers. It then sends the resulting packet to the SCION dispatcher for forwarding. The dispatcher takes care of sending the packets on the underlying network, and also forwards back return traffic based on the registered local address.

The application can close the connection once it is no longer needed. At this point, snet frees up any resources associated with the connection (e.g., by removing any active registrations from the dispatcher).

In addition to basic packet forwarding, snet also includes support for SCMP, thus allowing applications to explore the network or react to SCION network events. For example, applications can send pings to remote SCION hosts, or can react to SCMP error messages about on-path interfaces being temporarily down. Snet, in fact, does not react directly to a failed path, rather it expects the application on top to change path. We later present Nesquic (§12.2.2), a library that automatically handles network events.

### 12.1.4 SCION on Android

To demonstrate the portability of the SCION software components, we have developed a SCION app for Android,[3] which enables running an entire SCION AS on an Android smartphone. The SCION Android app makes the setup and configuration of a SCION AS on Android effortless. In its first release, the app supports pinging other SCION ASes and reading data from a SCION sensor server.

The latest app contains the SCION source code compiled into a binary that can be executed on Android. Every SCION component (dispatcher, daemon, etc.) runs in a separate process, which communicates with the SCION Android app via sockets, as is done on other platforms. This approach scales sufficiently well to run an entire SCION AS on an Android device, without the need to depend on an existing AS.

In the future, we anticipate that the SCION host components could be entirely integrated into the kernel.

## 12.2 Future Approaches

In the following, we present some approaches currently under development, which show how we envision the use of SCION by application developers in the future. First, we discuss the challenges in supporting applications to make optimal path choices in §12.2.1 and present the prototypical implementation

[3]The SCION Android app is available from the Google Play store at `https://play.google.com/store/apps/details?id=org.scionlab.scion`