Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Implementing and Evaluating MPTCP on the SCION Future Internet Architecture

Master Thesis

Michael A. Flückiger

August 25, 2020

Advisors: Prof. Dr. Adrian Perrig, Jean-Pierre Smith

Department of Computer Science, ETH Zürich

Abstract

This work relies on the two technologies MPTCP and SCION. Multipath TCP (MPTCP) is an extension to the Transmission Control Protocol (TCP). In contrast to TCP, MPTCP uses several sub-connections, so-called flows, for data exchange. An approach that has recently become increasingly popular, fitting the needs of today's multihomed devices. SCION is a secure internet architecture designed to address the weaknesses and shortcomings of today's Internet. It implements path transparency as an important feature. In contrast to the current Internet, SCION gives both, the sender and the receiver, control and knowledge of the paths along where their data is exchanged.

In this thesis, we present the implementation and evaluation of Shila, an approach to combine these two technologies. With this name-giving shim layer, the use of TCP applications over the SCION network becomes possible. Thanks to Shila, the large number of such TCP applications can be tested via SCION without the need to change its implementation. If hosts support MPTCP, one also benefits from its advantages and the inherent support of multiple paths in SCION. For example, Shila allows the paths for the individual MPTCP flows to be selected according to different criteria, such as being as short as possible.

Our implementation uses virtual network interfaces for the interaction between Shila and the applications. Created during startup of Shila, each virtual interface offers the possibility for a single flow to an MPTCP connection. For data exchange between Shila instances on different hosts, backbone connections via the SCION network are set up once a TCP connection is about to happen. If a client binds to one of the virtual interfaces to establish a new TCP connection, the IP traffic is intercepted by Shila. The SCION address of the host running the server is determined using the TCP address extracted from the received datagram and a hardcoded mapping. Shila contacts its counterpart on the receiving side via a dedicated endpoint listening at this SCION address and a well-known port. A main-flow, holding a backbone connection for data exchange, is established and linked to the TCP connection. MPTCP now starts to initiate further flows via each additional available virtual interface. Linked with its main-flow, Shila has all the information necessary to set up individual backbone connections for these sub-flows accordingly.

We have evaluated Shila in the SCIONLab network using iPerf3 as an exemplary application. The measurement has shown, that the throughput can be increased by using multiple paths. Compared to the implementation of QUIC via SCION, our approach performs worse. The detour through the kernel and Shila reduces the performance. Furthermore, the sending of redundant header information via the backbone connections causes an unnecessarily high overhead.

With the finally presented approaches to improve Shila this work lays the foundation for continuing development, improvement and research, which will also benefit the further deployment of SCION.

Contents

Chapter 1

Introduction

This chapter provides an easily accessible introduction to the subject covered by this thesis. First, the main ingredients, namely Multipath TCP and SCION, are shortly introduced. Afterward, we explain how this work combines these two concepts with a shim, intermediary layer, and why this combination could be interesting. Every subject mentioned in this introduction is covered in more detail in the report's subsequent parts.

1.1 Transmission Control Protocol (TCP)

TCP stands for Transmission Control Protocol, a reliable, connection-oriented transport protocol in computer networks. It is one of the most important protocols and forms the basis of today's Internet. Connection-oriented means, that for every data transport between two entities, a fixed connection between the two endpoints is established. During all the data exchange, this fixed connection is then used, while none of the endpoints can influence the path the data takes through the network. As an example, imagine a user wants to surf its favorite web page: Upon starting browsing, TCP creates in the background a fixed connection between the user computer and the server hosting the page, illustrated in Figure 1.1. All data sent and received by the client's browser runs through this established connection.

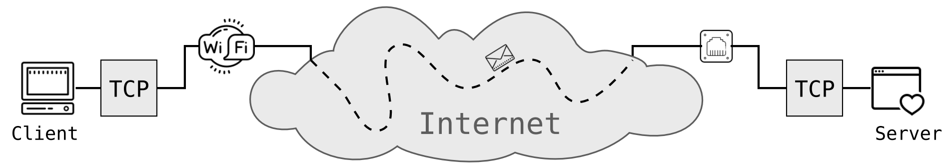

Figure 1.1: Simplified illustration of TCP where the data exchange happens along a single fixed connection. The choice of the path taken cannot be influenced by the endpoints, in the illustration indicated by the dashed line.

1.2 Multipath TCP (MPTCP)

MPTCP stands for Multipath TCP and is an extension to TCP. It works exactly like TCP, with the only difference that all available network interfaces of a computer are used for the connection. This means, that when the user visits his favorite website, MPTCP creates not just one connection, but one sub-connection, a so-called flow, for each interface available. Nowadays devices often have multiple attachment points to the Internet. MPTCP creates, as illustrated in Figure 1.2, one flow via WLAN, one via cable and maybe even one via an analog dial-up modem. Thanks to these multiple flows, the protocol can improve reliability and performance. However, also with MPTCP, endpoints cannot influence what paths are taken by the individual flows.

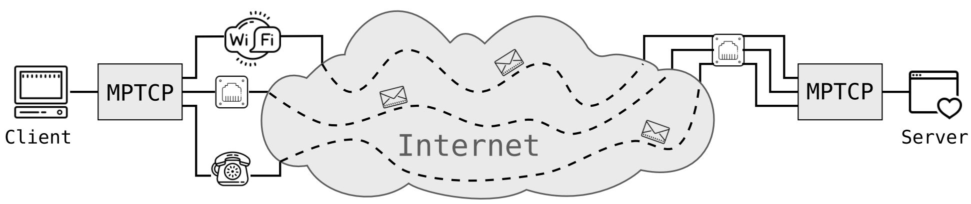

Figure 1.2: Simplified illustration of MPTCP, where the endpoints use all available network interfaces for the data exchange. As for TCP, the choice for the paths taken cannot be influenced by the user.

1.3 SCION

Scalability, Control and Isolation on next-generation Networks or SCION for short is a new type of internet architecture, which has been developed by ETH Zurich for several years. SCION re-implements existing Internet protocols taking known vulnerabilities into account. It is more resource-efficient and reduces the supremacy of individual actors through its decentralized infrastructure. An important feature of the architecture, illustrated in Figure 1.3, is that endpoints can determine the path taken by its data through the network. Senders can choose the path along which they want to send their data, receivers can choose from which paths they want to receive data.

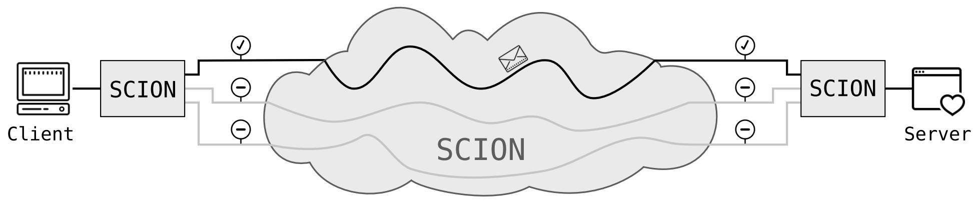

Figure 1.3: Simplified illustration of the SCION internet architecture where the endpoints can determine which paths their data takes through the network.

1.4 Shila

Functionality

Shila is the abbreviation for shim layer and its implementation and evaluation is the main contribution of this work. It acts as a mediating layer and enables the use of MPTCP in the SCION internet architecture; illustrated in Figure 1.4.

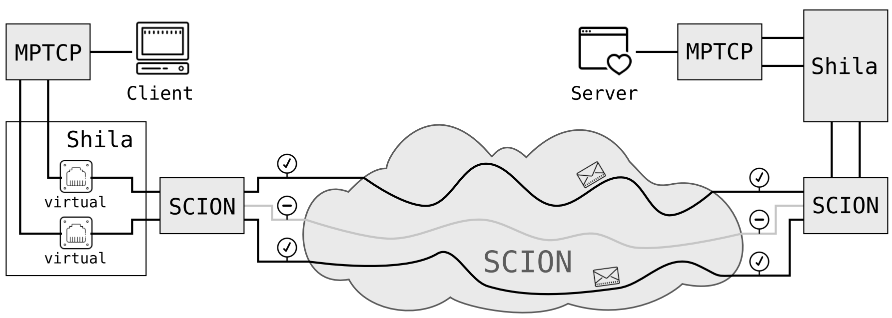

Figure 1.4: Simplified illustration of the functionality of Shila: It acts as an intermediate layer between MPTCP and SCION, provides virtual interfaces to MPTCP and redirects the intercepted flows through the SCION network.

Upon startup, Shila creates and monitors a user-selectable number of virtual network interfaces. A virtual interface is a fake connection to a network, which is perceived by the computer as a real one. If now someone visits his favorite homepage, Multipath TCP tries to connect to the server via these virtual interfaces. Shila detects this connection establishment and redirects the data. For every flow created by MPTCP, the Shila instance creates a connection over SCION to the Shila instance running on the server and forwards the respective data traffic. The shim layer on the server side receives the data and distributes it through the MPTCP functionality to the server application.

Benefit

The new internet architecture SCION offers great advantages over the existing Internet in terms of security and reliability. However, the transfer from such an established infrastructure as the current Internet to new technology is a big challenge. Applications rely on established programming interfaces and protocols and are programmed accordingly. A re-implementation of all software so that it can be used in a new architecture seems utopian. For the deployment of SCION, it is therefore of great importance that existing applications can be used in the new network with as little effort as possible. The implementation of Shila enables exactly this. All applications that use TCP for their communications can be operated via SCION without the need for internal changes. With Shila, your favorite TCP application can be run over SCION without much effort - a good reason to try the new infrastructure.

If the endpoints furthermore support MPTCP, the application can additionally benefit from the advantages of the multipath protocol and the support of multiple paths in SCION. But not only that, with Shila in its mediating role in between, there is further potential. To give a possible example, let's look at the use of MPTCP in today's Internet. It is quite possible that several flows of an MPTCP connection use the same link for certain parts of the path on the Internet. In order not to disadvantage other users, a fairness algorithm[1] monitors and controls the use of bandwidth. This is to ensure that all flows of an MPTCP connection going through the same link do not have more bandwidth available than the connection of another user. As we have seen, the endpoints on the Internet today are unaware of the paths their data takes. The same applies to the fairness algorithm, it has also no idea what paths are taken and can therefore not tell whether shared sub-paths exist or not. It has to act conservatively. If we use Shila to run our applications with MPTCP over the SCION network, Shila could ensure that no links are shared when choosing the paths of the individual flows. This makes it possible to disable the fairness algorithm or at least make it less strict, which results in better performance of the data transfer.

[1]Ensuring fairness is part of Congestion Control. However, to keep the introduction as generally understandable as possible, we don't use this term here.

Chapter 2

Related Work

In this chapter, we provide the reader with points of reference for related work and technologies.

A detailed review of the development of multipath transport protocols gives the survey [14]. It discusses reasons for the paradigm shift from single to multipath and the challenges of this transition. Furthermore the Stream Control Transmission Protocol (SCTP) [46, 39] and Multipath TCP (MPTCP) [2, 41, 20, 11], the two most famous protocols in these area, are investigated.

SCTP shares several characteristics with TCP [37]. Both protocols are reliable, connection-oriented and use a rate-based mechanism for Congestion Control. Unlike with TCP, with SCTP a connection can consist of multiple streams, either used in a multiplexed fashion or simultaneously. This prevents the protocol from suffering from the Head-of-line blocking problem [42] which is an issue with TCP. SCTP furthermore supports multihoming, a feature originally only intended to be used for failover. However, works like Concurrent multipath transfer SCTP (CMT-SCTP) [21] or wireless multipath transfer SCTP (WiMP-SCTP) [16] propose efforts to benefit from the availability of multiple paths by using them concurrently. The Stream Control Transmission Protocol appeared when TCP had already established itself. Making a change to SCPT would require adjustments to the applications and network stack, the most important factors why the protocol was not successful [15].

Multipath TCP (MPTCP), as an extension to TCP, is the most recognized approach to multipathing. It can be used by all TCP applications without any changes to their implementation. With MPTCP, multiple paths are used for the data exchange within a single connection. This functionality for example provides advantages in the field of smartphone use. The protocol facilitates the combination of WLAN and cellular access [5] or allows fluid handovers between individual connection points [35]. Well-known smartphone vendors such as Apple, LG or Samsung rely on Multipath TCP in their devices for the pleasant switch between 4G and Wi-Fi [4]. In addition to mobile phones, for example also data centers benefit from better performance thanks to MPTCP [40].

In the work [6], the implementation of an extension for QUIC [22, 38], called Multipath QUIC (MPQUIC), is presented. QUIC itself is a relatively recently developed protocol running on top of UDP. Similar to SCTP, QUIC supports several streams but not the use of more than one path. MPQUIC enables the protocol to use multiple paths. An in-depth discussion of (MP)QUIC and an investigation of MPTCP with the focus on use cases with smartphones, devices predestined for multi-homing, is given in this thesis [7].

Increased demands on reliability and performance as well as the availability of multiple network interfaces have led to the development of the mentioned multipath protocols. Although the Internet was not originally designed for the use of multiple paths, it allows it. This is contrary to new internet architectures such as NEBULA [1] or SCION [49, 36], where multipathing is an inherent part of the design and a cornerstone of security and reliability.

The TUN/TAB drivers [25] provide accessibility of virtual network devices to applications in userland. This functionality is mainly used for tunneling, for example by VTun [26] or OpenVPN [19].

Another approach to access network traffic in userland is XDP [17, 48], short for eXpress Data Path. It is not the first technology that allows the programmable processing of packets, but its approach differs from known solutions. For performance reasons, known products like the DataPlane Development Kit (DPDK) [12] bypass the kernel with all its important security and isolation mechanisms at all. This stays in contrast to XDP where programmable packet processing is integrated directly into and verified by the kernel. A customizable XDP program is executed as part of the device driver on every packet. After parsing and modification within this so-called XDP driver hook, a packet can be dropped, pushed to the kernels network stack or redirected. One possible destination for redirection is AF_XDP [30, 23, 8]. AF_XDP is a special socket type that allows the exchange of raw packets between the network interface and a userland application at a high speed.

Chapter 3

Important Concepts

In this chapter, we present the most important building blocks on which this work is based. The goal is to give the necessary insights to allow a self-containing consumption of this thesis and place the work in a larger context. The three building blocks are the Transmission Control Protocol (TCP), the Multipath TCP (MPTCP) based on it and the new internet architecture SCION. These three concepts are summarized in the corresponding order in the following sections.

3.1 Transmission Control Protocol (TCP)

This recap about the widely used transport protocol TCP is mainly inspired by the summary [27].

The Transmission Control Protocol (TCP) is one of the main protocols in today's Internet. Its two main developers Vint Cerf and Bob Kahn started working on the protocol in 1974. The original specification was written in 1981; defined in the RFC document number 793 [37].

TCP is a byte stream, connection-oriented and reliable transport protocol. In the OSI model, it is situated between the application layer above and the network layer below. TCP receives the data in a byte stream from the application. It is up to the protocol to partition this stream into so-called TCP segments in order to transmit the data in manageable pieces to the receiver. Before two endpoints can exchange data over TCP, they must first agree upon the willingness to communicate; hence connection-oriented. Similar to a call with a telephone, first a connection has to be established before the two parties can exchange information. We describe the procedure of a TCP connection establishment and termination later in this section. To ensure reliable data transfer TCP uses different mechanisms. A per-segment checksum enables the receiver to detect crudities in the transferred data. Everything that is received multiple times, is discarded by the receiving side. It is also the receiver that ensures that the received data is transferred to the application in the correct order. TCP also has a retransmission functionality that ensures delivery of the data in the case of data corruption or loss. The successful reception of data is confirmed by the receiver with a positive acknowledgment. If the sender does not receive such positive acknowledgments for a certain period of time, it will resend the data.

TCP Segment

TCP partitions the byte stream into TCP segments consisting of a header and a data part. Figure 3.1 depicts a valid format of a TCP segment with its different fields. In the following, the fields most important for the understanding of TCP are described in more detail.

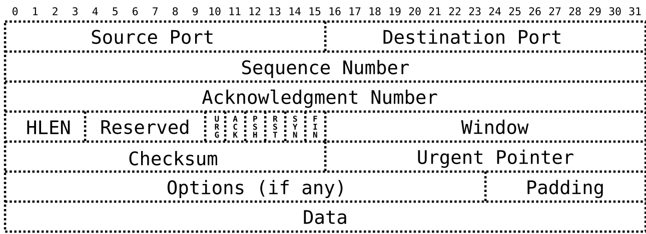

Figure 3.1: Illustration of a TCP segment.

Source Port: 16-bit number tagging the application where the segment originated from within a sending endpoint.

Destination Port: 16-bit number tagging the application where the segment is destined for on a receiving endpoint.

Sequence number: 32-bit number identifying the current position of the first data byte in the segment within the entire byte stream for the TCP connection.

Acknowledgment Number: 32-bit number telling the receiver which data byte the sender is expecting next. It is always one greater than the most recently received data byte and only valid if the ACK control bit is set.

ACK (Control Bit): If set, the acknowledgment number is valid.

Window: 16-bit number used by TCP for flow control. It specifies the number of bytes the sender of this segment is currently willing to receive.

SYN (Control Bit): Sender and receiver use this bit during the connection establishment to synchronize their sequence numbers.

FIN (Control Bit): This bit is part of the connection termination. It is set by the sender if its byte stream has reached the end.

Options: Provides space for additional optional parameters that one may want to exchange between sender and receiver.

Note that a single TCP connection is uniquely identified by a complete pair of IP addresses (source and destination) together with a complete pair of TCP ports (source and destination).

Connection Establishment, Data Transfer and Termination

TCP is connection-oriented, meaning that there is a virtual connection between two endpoints exchanging data. Such a virtual connection runs through three stages: connection establishment, data transfer and connection termination.

Connection Establishment Two hosts, A and B, willing to communicate establish a connection by exchanging a predefined set of messages known as the Three-Way Handshake. Figure 3.2 on the left side illustrates this sequence.

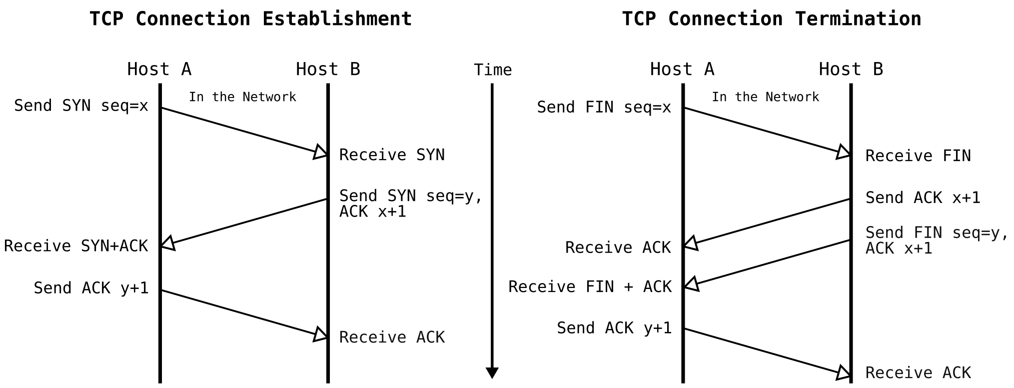

Figure 3.2: Illustration of the TCP connection establishment (l.) and TCP connection termination (r.) messages exchange.

Host A as an initiator of the connection sends a segment containing an initial sequence number x and the SYN control bit set to Host B.

Host B processes the received segment and answers with a segment of its own. It contains an initial sequence number y and has the SYN control bit set. Furthermore Host B assigns x+1 to the acknowledgment number and sets the ACK control bit. This indicates that the next byte expecting from Host B should contain data starting with sequence number x+1.

Host A finishes the connection establishment phase upon reception of Host

B's SYN+ACK. It sends a last acknowledgment to Host B, indicating that the next expected byte should start with sequence number y+1.

Data Transfer Once the Three-Way Handshake is completed, the two hosts can exchange data with each other. Important functionalities of the data transfer are flow control and congestion control. These concepts are discussed later, first the focus lies on some key ideas.

In a basic TCP implementation, segments are put into the network by the sender as long as there is data available and as long as the window announced by the receiver is not exceeded. The receiver consumes the segments and acknowledges them by sending back positive acknowledgments together with its current position in the byte stream. The announcement of the current window size is also included in these messages sent back. If data gets lost or duplicated, a hole may exist in the byte stream. The receiver continues to acknowledge the most current contiguous position in the byte stream accepted so far.

The sender must halt transmission in the case where data ready to be sent will exceed the receivers advertised window size. Upon reception of further positive acknowledgments announcing a window size greater than zero, the sender can continue delivering the data.

If there is no data to send, TCP waits patiently until either there is some again or the other end of the connection sends data to receive.

Connection Termination A TCP connection is completely terminated by the exchange of four segments as depicted on the right side of Figure 3.2. Since the protocol is full-duplex, both ends must be torn down independently. To initiate this process the FIN control bits are used.

As soon as the application running on Host A wants to close the connection, a FIN segment is sent from Host A to Host B. Upon reception of this segment, Host B acknowledges it and notifies its destination application about the termination request. As soon as this application shuts down the connection as well, Host B sends out a FIN segment which is finally acknowledged by Host A.

Flow Control

Flow control is an important part of the data transfer in TCP. Its purpose is to properly match the transmission rate of the sender to the one of the receiver and the network. For performance reasons, one is interested in a transmission rate that is high enough without overwhelming the receiver or the network. TCP uses the window field to adjust the rate of flow between TCP endpoints. This concept is called sliding window. In every segment, the receiver specifies with the window field how much data it is willing to buffer. The sender can only send up to that amount of data before it has to wait for further acknowledgments and window updates from the receiver.

Congestion Control

Another major part of the data transfer in TCP is congestion control. However, this term is misleading, since it rather should be called congestion avoidance. TCP itself cannot control the congestion, it rather tries to avoid congestion using different mechanisms, which are briefly discussed in the following.

Slow Start Slow start is a mechanism used by the sender to control the sending rate. Hereby, the rate of acknowledgments returned by the receiver controls the rate at which the sender can transmit data. At the very beginning of a TCP connection, the congestion window is initialized to one. With every positive acknowledgment received, the congestion window is increased by one. The sender can send the minimum of this congestion window and the advertised window of the receiver, denoted as transmission window. Unlike the name suggests, slow start is by no means slow. In a congestion-free network with good response time, the congestion window doubles for every round-trip-time. The increase of the congestion window will continue until the maximum transmission window is reached or until congestion finally occurs.

Congestion Avoidance The congestion avoidance algorithm has two mechanisms to detect network congestion. If the blockage is indicated through the reception of duplicate acknowledgments, the transmission window is halved and the fast retransmit and fast recovery mechanism take over. If the congestion was indicated through a timeout of the retransmission timer, the sender is put back into slow start mode, i.e. the transmission window is set to one. After running into congestion, the congestion window is again increased using slow start. However, it is only used up to the halfway point where congestion originally occurred. After reaching this point, the congestion window is just increased by one for all segments in the transmission window that are acknowledged. The linear growth from this point onward makes sure that TCP increases its transmission rate more slowly towards the previous congestion point.

Fast Retransmit If the sender receives an acknowledgment multiple times, it is not clear if it is because a segment was lost or simply because a segment was delayed and received out of order at the receiver. If the latter is the case, it should not take too long until the latest expected acknowledgment arrives. Typically, for simple out of order condition, not more than one or two duplicated acknowledgments should be received. More duplicates are a strong indication that a segment was lost. Upon reception of three duplicated acknowledgments, the sender does not wait until the retransmission timer expires but resends the segment immediately and enters fast recovery.

Fast Recovery Duplicate acknowledgments can only be generated when a segment is received at the receiver. Therefore, as long as the sender receives acknowledgments it knows that there is data flow through the network. The loss of a segment was probably just a rare event, instead of reducing the flow drastically by using slow start again the sender just enters congestion avoidance without decreasing the size of the congestion window too much. This behavior allows for higher throughput in cases of only moderate congestion.

Conclusion

TCP is a rather complex protocol that handles an enormous amount of functionality in today's Internet. The presented discussion gives only a small insight into the scope of TCP. However, it provides the necessary insights to understand the extensions to TCP, namely Multipath TCP, discussed in the next section.

3.2 Multipath TCP (MPTCP)

In this section, we introduce Multipath TCP. The presented summary is based on [2, 41, 47] and the RFC document number 6824 [11].

Basics

In times where most hosts are equipped with multiple physical interfaces, a protocol that binds each connection to a single interface does not exhaust the potential of the infrastructure. TCP is not capable of using the multiple paths to the Internet available for an increase in connection redundancy and performance. Multipath TCP, as an evolution of TCP, is an attempt to utilize the multiple paths available to gain robustness and performance advantages.

Multipath TCP allows an unmodified application to start what it believes a regular TCP connection using the well-known API for TCP. If both endpoints support the use of MPTCP, multiple flows are established under the hood. The data is shared among these flows, where most data is sent out on the least congested path. If the use of MPTCP is not supported a regular TCP connection is established instead. Multipath TCP works in all scenarios where TCP currently works. By using an appropriate congestion control MPTCP furthermore ensures that other, regular TCP connections present in the network are not disadvantaged.

Connection Setup

MPTCP uses the TCP Three-Way Handshake to set up the initial connection, i.e. the main-flow. The sender includes the MP_CAPABLE option in the SYN, which is also included in the returning SYN+ACK when the server is ready to use MPTCP. The sender concludes the Multipath TCP connection setup by including the MP_CAPABLE option in the third packet of the handshake as well. The connecting entities fall back to regular TCP if not all segments being exchanged as part of the Three-Way Handshake contain the respective option. The MP_CAPABLE option itself contains a 64-bit key generated by the sender and the receiver, later used to verify the authenticity of additional flows. The sequence of segments exchanged for an MPTCP connection establishment between two Host A and B is illustrated in Figure 3.3.

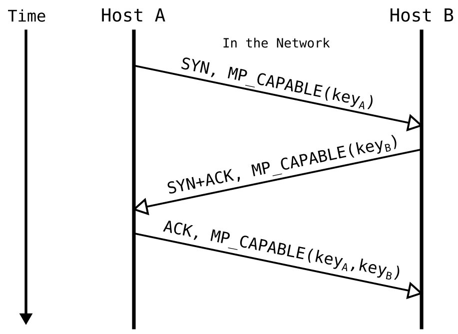

Figure 3.3: Illustration of the sequence of segments exchanged for the MPTCP connection setup. The MP_CAPABLE option has to be included in all messages exchanged for the establishment to succeed.

Addition of Sub-flows

Once a connection, i.e. the main-flow, has been established, further flows, so-called sub-flows, can be added. For this, MPTCP again performs a regular TCP Three-Way Handshake using addresses or ports of additionally available interfaces. To add the sub-flow, the MP_JOIN option has to be included in any segment part of the handshake. This option holds the message authentication code (MAC) of the keys exchanged in the previously described initial connection setup. This measure makes it difficult for malicious third parties to hijack an existing connection. Furthermore, the option includes a connection identifier, derived as a hash from the receiver's key, used to match new sub-flows with an existing connections main-flow.

Data Transfer

Data exchanged between client and server can be sent over any flow part of a Multipath TCP connection. To remain a reliable protocol, MPTCP must retransmit data over a different flow if a certain flow fails. This is achieved by making use of two different principles discussed in the following and illustrated in Figure 3.4.

Flow and Data Sequence Number In Multipath TCP every flow is equivalent to a regular TCP connection using its own 32-bit sequence number space. With this flow sequence number, the current position within the entire byte stream transferred over a certain flow is identified. Additionally, MPTCP maintains a 64-bit data sequence numbering space, which is used by the DSS_MAP and the DSS_ACK option. The DSS_MAP contains a mapping between the flow sequence number and the data sequence number. It is used by the receiver for correctly ordering the byte stream, possibly received disordered over the different flows, for the application.

Two different Acknowledgment Levels The acknowledgment of received data happens on two different levels. The regular TCP acknowledgment number is used to confirm the reception of data on a flow connection level. Standard retransmission mechanisms, as discussed earlier, are used to recover from a segment loss indicated through the reception of multiple acknowledgments confirming the same flow sequence number. With the additional DSS_ACK returned by the receiver as part of the ACK's, cumulative acknowledgments at a data sequence level are available. These allow MPTCP to detect the failure of a flow and to initiate the retransmission of lost data over a different flow.

Congestion Control

One way to implement congestion control in MPTCP is to use conventional mechanisms on a flow connection level. This means that each TCP flow maintains its congestion window without affecting or being affected by other flows and their window. Possible congestion control could be any suitable for TCP, like the default AIMD approach [29] mentioned in the previous section or derivatives like CUBIC [18]. But with independent congestion windows for each flow, other users in the network are disadvantaged. Imagine that several flows of an MPTCP connection and a single external TCP connection share the same link. In this case, the MPTCP connection may use more bandwidth than it is entitled to. This is possible because each flow is treated as a single, independent TCP connection. To eliminate this injustice MPTCP uses a congestion algorithm which makes the windows of the individual flows dependent on each other. One possible implementa-

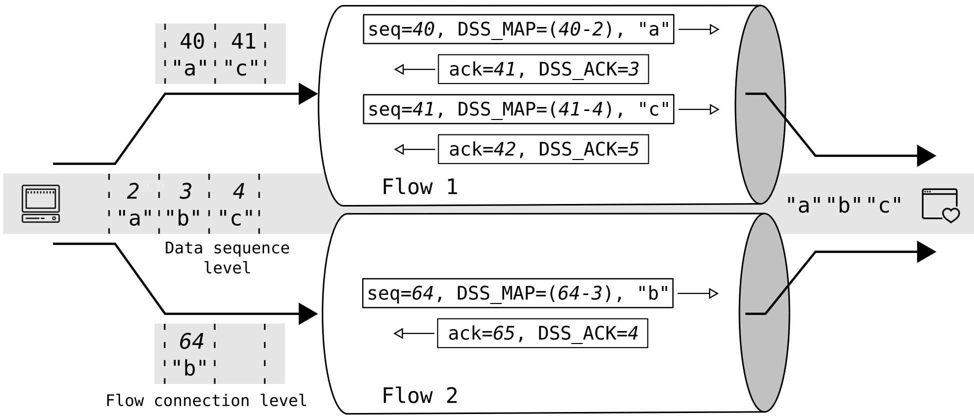

Figure 3.4: Illustration of data transfer in MPTCP with two flows. The regular TCP sequence numbers are used to acknowledge data reception and trigger mechanisms for re-transmission on a flow connection level. With the additionally introduced DSS_ACK's on a data sequence level, MPTCP is able to detect dying flows and can maintain the ordering of exchange data.

tion hereby is the Linked Increase Algorithm (lia) [3], initially proposed by the community. In the meantime, further developments, for example the Opportunistic Linked Increase Algorithm (olia) [24] or the Balanced Linked Adaptation Congestion Control Algorithm (balia) [44] are available.

Conclusion

The benefits of MPTCP as an extension are promising, as already mentioned in Chapter 2, especially in the area of smartphone usage. With this summary, we have by no means covered all facets of Multipath TCP. But it does provide a basis for understanding this work and how we try to benefit from the protocol as well. An overview of the current development around MPTCP can be found here [20].

3.3 SCION

In the upcoming summary, we introduce the fundamental concepts of SCION. It is based on the SCION book [36] and a selection of videos [34]. More information and the latest news about the development of SCION can be found on the official web site [32].

Shortcomings of today's Internet

We start by mentioning some of the Internet's shortcomings, more precisely the shortcomings of two important building blocks of it. Today's Internet is of great importance for the economy, politics and society worldwide. Despite this importance, the global infrastructure does not meet the requirements in terms of security and availability to any degree. The core of the Internet is based on a few protocols developed almost 30 years ago. The Internet Protocol (IP) and the Border Gateway Protocol (BGP) control most system-relevant functions, but have undergone only minimal development since their deployment and suffer from several shortcomings.

Internet Protocol (IP) and its Shortcomings IP handles the transport of packets between end-users along a single path through the Internet. It routes them from a starting point to a destination, the endpoints have no influence on or knowledge of the path the packets take. All units involved in the routing determine the next hop based only on the destination address and a routing table. For IP, one can note the following shortcomings: The protocol does not sufficiently separate the functionality of routing and forwarding, sudden updates of routing tables can adversely affect a path, e.g. change its direction or even break it. Furthermore, the protocol does not offer the end-user with the possibility to control or check the path his data takes through the Internet. This functionality would be desirable in many situations, e.g. allowing a sender to avoid sending its data through malicious nodes. Furthermore, the IP protocol is based on the use of routing tables. The high-performance lookup of next-hop destinations is energy-intensive and the routing tables can be maliciously exhausted.

Border Gateway Protocol (BGP) and its Shortcomings BGP is responsible for the exchange of routing information between independently operating autonomous systems, denoted as ASes, like Internet service providers (ISPs). These providers use BGP update messages to announce the IP prefixes which are reachable through their subnet. Each ISP can decide based on its criteria, which announcements it wants to share further and which routes it wants to use. This traffic engineering is mainly driven by economically influenced policies. BGP also suffers from some shortcomings. It may take a considerable amount of time for the routing infrastructure to return to a stable state after new announcements have been made. Route changes directly affect forwarding and can lead to failures and interruptions of the data exchange. Furthermore, notifications affect all BGP entities that are present in the network. There is no hierarchy or isolation between individual areas. A single faulty BGP entity can severely disrupt the entire global routing infrastructure. Finally, BGP only allows one to select a single path for the routing between different autonomous systems.

SCION - Scalability, Control and Isolation on Next-generation Networks

Intending to remedy the above-mentioned shortcomings and create a better, safer internet, the development of SCION started with the publication of the first paper [49] in 2011. SCION, an acronym for Scalability, Control and Isolation on Next-generation networks, shall become a secure network architecture that provides high availability regardless of the presence of malicious actors and offers efficient point-to-point packet delivery. In the ensuing section, we give an introduction to the correspondingly implemented SCION architecture. Its implementation and design are guided by the following objectives.

High Availability: The functionality of the network is guaranteed even in the presence of a malicious party as long as at least one benign path exists between communicating endpoints.

Path Transparency: Endpoints have control over and insight into the path their data takes through the network.

Plane Separation: There is a strict separation of control plane and data plane which ensures that their functionality is not influenced by each other.

Multipath Support: The architecture allows an endpoint to send data to its destination along multiple paths.

Router State Avoidance: Routing works without the routers having to store a state.

The SCION Architecture

The SCION architecture combines autonomous systems (ASes) into so-called isolation domains (ISD). Each ISD is managed by a subset of ASes, which form the ISD core and are denoted as core ASes. These core ASes agree on a policy, a trust root configuration (TRC), which applies inside their ISD and is used to validate bindings between names and public keys or addresses. Mapping to the real world, an isolation domain can for example represent a single state or the union of several states with the same legal understanding. Figure 3.5 depicts a possible grouping of ASes into different ISDs connected by respective links.

To enable an exchange between two endpoints, one or more paths through the network must be defined along where the data is sent. Every single step which is necessary to create such forwarding paths can be assigned to one of two planes. The control plane is responsible for discovering the paths and making them available to the end hosts, whereas the data plane is responsible for forwarding the data along the corresponding paths. This strict separation of routing (control plane) and forwarding (data plane) is one of the key features of SCION. The individual steps of creating a forwarding

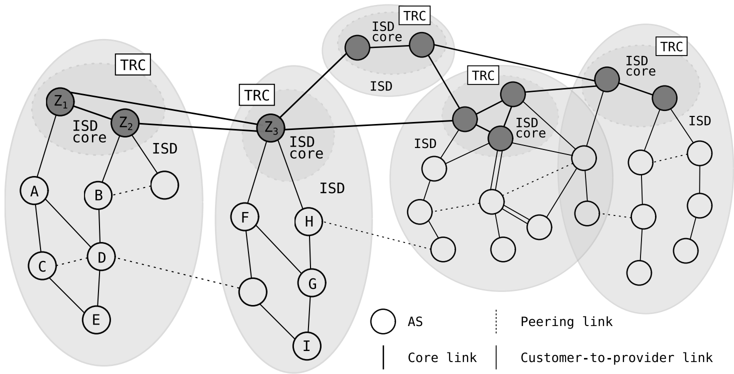

Figure 3.5: Illustration of multiple ISDs with their ASes connected through different types of links. Core ASes via core links and ASes not belonging to the core via customer-to-provider or peering links. This drawing is inspired by the one shown in the SCION book [36] on page 18.

path are summarized below and illustrated in Figure 3.6. The heading of the individual parts indicates their assignment to the corresponding plane.

Path Exploration (Control Plane) The first step in creating a forwarding path is path exploration, called beaconing in SCION terminology. The core ASes regularly send path-segment construction beacons (PCBs) to all its neighbors. Upon reception, an AS adds its own information and forwards the PCBs further to its neighbors. This results in a sequence of ASes representing a path through the SCION network.

Path Registration (Control Plane) An AS typically receives several PCB's which describe path segments to different core ASes. It is up to each AS to decide which of the these it wishes to use. In the path registration, an AS registers the paths via which it wishes to be reached. For this purpose, the corresponding paths are registered at the path servers of the ISD, i.e. they are announced and thus available for the formation of forwarding paths.

Path Lookup (Control Plane) If a source host wants to create a connection to a target host, it carries out a path lookup. From the path servers in the network, it receives path segments that can then be combined, as part of the data plane functionality, into a forwarding path in a final step.

Path Combination (Data Plane) After a host has received individual path segments in the course of the path lookup, it can now combine these into a path to the destination. Depending on where the destination is located, up to three path segments must be combined. If the destination AS is on the

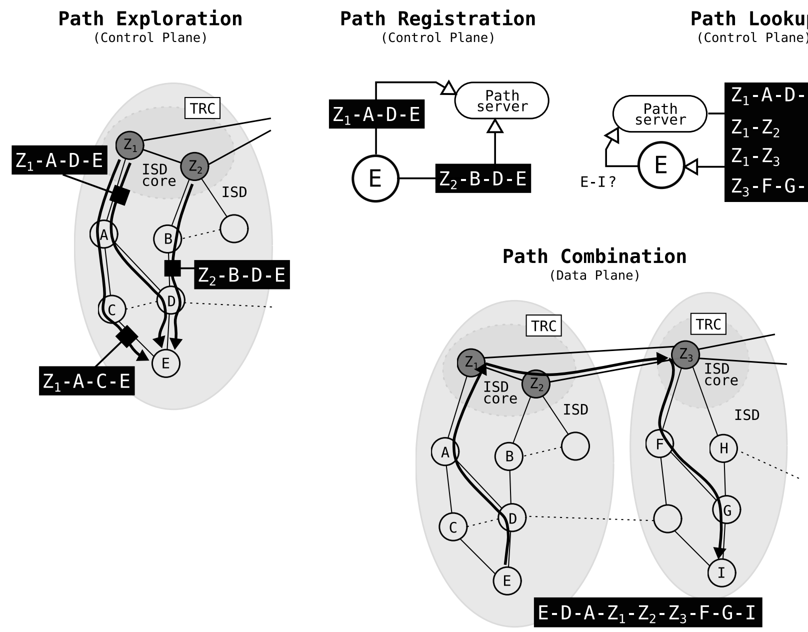

Figure 3.6: Illustration of the different steps involved in the creation of a forwarding path: During path exploration, the AS E receives three PCB's describing different paths to reach the core. It decides to register two of these at the path server. If AS E wants to reach AS I it performs a lookup at the path server to get possible path segments. These individual path segments can then in the path combination be arbitrarily joined to form a forwarding path used for data exchange.

path from source AS to its core AS, then one segment is sufficient. If path segments have an AS in common, two segments are necessary, namely one from the source AS to the shared AS and one from there to the destination AS. Three path segments are necessary if the path segments from source AS and destination AS to their respective core AS do not intersect. In addition to the path segments connecting the ASes to their core, a segment between the two core ASes is necessary.

Forwarding (Data Plane) Once a path is formed, it is included in the SCION packet header and used for forwarding. The recipient of a packet can send back data by simply inverting the path along where it got the packet or create a different one by performing a path lookup and combination on its own.

Conclusion

This summary only scratches the surface of the functionality that SCION provides. The new internet architecture offers a variety of additional functions, security mechanisms and extensions which are not mentioned here. The goal of this section was to explain why the transformation to a new internet architecture is so important and to introduce SCION as a promising candidate. Interested readers are again referred to the SCION book [36] or website [32].

Chapter 4

Shila - Introduction

This chapter introduces the central point of the thesis and the author's main contribution towards the establishment of a more secure internet architecture. A piece of software that allows the usage of MPTCP over SCION. The chapter includes a presentation of the software, an explanation of how to use it and a rough outline of the functionality under the hood. A detailed discussion of the implementation follows in Chapter 5.

4.1 Introduction

Shila could very well be the name of the female lead role in a thrilling crime novel. However, in the context of this work Shila is the abbreviation of shim layer. It appropriately names the implementation which, by mediating between the two technologies, enables the use of MPTCP over SCION.

Initial Situation

To simplify the following descriptions, the starting point for the use of Shila is defined. To use Shila, one needs at least two devices[1] with MPTCP installed. In addition, the devices must be connected through the SCION network. Guidance to install MPTCP can be found here [20], the setup of SCION is described in detail on the official SCION webpage [32].

Each device is identified by a SCION address, not surprisingly referred to as the SCON address of a host. TCP applications, or TCP endpoints, which are executed on a host are bound to a network interface. They are uniquely addressable through an IP address paired with a TCP port number, denoted as TCP address of an application. The individual parts of this tuple are called the IP address or the TCP port of an application. Furthermore, TCP

[1] The usage of multiple instances of Shila on a single device is also possible. For illustrative purposes, however, this scenario is not discussed here.

applications can also be linked with addresses of the SCION namespace. The same principle applies here. Thus, an application may also be identified either by a full SCION address or just individual parts of it, e.g. just the SCION port.

Usage

In the following, a simple example case describes how a user can utilize Shila. A description on how to use Shila is also part of the Shila code repository [10].

As an example, a setup with two endpoints is considered. Host 1 with SCION address 1-ff00:0:112,127.0.0.1 and Host 2 with address 2-ff00:0:220,127.0.0.1, both are part of the SCION network. The goal is to establish a connection between the client instance of an TCP application running on Host 1 and the respective server instance with TCP address 10.7.0.9:27041 running on Host 2. As an example TCP application, we consider iPerf3, the well-known measurement tool.

First, the so-called Routing-Information for the Shila instance running on Host 1 has to be prepared. It describes the mapping between TCP address of an application running on a certain host and the same host's SCION address. The routing information stored in a file named *routing.json* and is located in the Shila root directory. For the given example it contains only one entry:

```
[{ "key"  : { "ip" : "10.7.0.9", "port" : "27041" },
   "flow" : { "address"  : "2-ff00:0:220,127.0.0.1:27041" } }]
```

In the second step, the Shila instance on both hosts can be started. If no arguments are specified the program starts with the default settings and loads the routing entries from the previously mentioned file:

```
sudo ./shila
```

The server instance of iPerf3 can be started on Host 2 once the shim layer is running. To be reachable, the instance has to be started in the ingress network namespace of Shila:

```
sudo ip netns exec shila-ingress iperf3 -s -p 27041
```

As final step the client part of iPerf3 is executed in Shilas egress network namespace on Host 1:

```
sudo ip netns exec shila-egress iperf3 -c 10.7.0.9 -p 27041
```

```
Host 1 - Starting Shila..

INFO: Shila up and running.
INFO:
INFO: MainFlow AZHX - 1 connection successful established!
INFO: | TCP-Flow:        21.199.102.49:35384 <-> 10.7.0.9:27041
INFO: | Net-Flow:        127.0.0.1:3354 <-> 1-ff00:0:111,127.0.0.1:27041
INFO: | Metrics:         1280 (mtu) 4 (length)
INFO: | Sharability:     0
INFO: | Main-Flow:       {{21.199.102.49 35384 } {10.7.0.9 27041 }}
INFO: | Hops: [1-ff00:0:112 1>2 1-ff00:0:110 1>41 1-ff00:0:111] MTU: 1280, NextHop: 127.0.0.25:31020
INFO:
INFO: SubFlow AZHX - 2 connection successful established!
INFO: | TCP-Flow:        84.132.127.112:51637 <-> 10.7.0.9:27041
INFO: | Net-Flow:        127.0.0.1:3358 <-> 1-ff00:0:111,127.0.0.1:27041
INFO: | Metrics:         1280 (mtu) 4 (length)
INFO: | Sharability:     0
INFO: | Main-Flow:       {{21.199.102.49 35384 } {10.7.0.9 27041 }}
INFO: | Hops: [1-ff00:0:112 1>2 1-ff00:0:110 1>41 1-ff00:0:111] MTU: 1280, NextHop: 127.0.0.25:31020
INFO:
INFO: MainFlow AZIF - 1 connection successful established!
INFO: | TCP-Flow:        21.199.102.49:35392 <-> 10.7.0.9:27041
INFO: | Net-Flow:        127.0.0.1:3360 <-> 1-ff00:0:111,127.0.0.1:27041
INFO: | Metrics:         1280 (mtu) 4 (length)
INFO: | Sharability:     0
INFO: | Main-Flow:       {{21.199.102.49 35392 } {10.7.0.9 27041 }}
INFO: | Hops: [1-ff00:0:112 1>2 1-ff00:0:110 1>41 1-ff00:0:111] MTU: 1280, NextHop: 127.0.0.25:31020
INFO:
INFO: SubFlow AZHX - 3 connection successful established!
INFO: | TCP-Flow:        214.41.122.79:56285 <-> 10.7.0.9:27041
INFO: | Net-Flow:        127.0.0.1:3359 <-> 1-ff00:0:111,127.0.0.1:27041
INFO: | Metrics:         1280 (mtu) 4 (length)
INFO: | Sharability:     0
INFO: | Main-Flow:       {{21.199.102.49 35384 } {10.7.0.9 27041 }}
INFO: | Hops: [1-ff00:0:112 1>2 1-ff00:0:110 1>41 1-ff00:0:111] MTU: 1280, NextHop: 127.0.0.25:31020
INFO:
INFO: SubFlow AZIF - 2 connection successful established!
INFO: | TCP-Flow:        214.41.122.79:51425 <-> 10.7.0.9:27041
INFO: | Net-Flow:        127.0.0.1:3364 <-> 1-ff00:0:111,127.0.0.1:27041
INFO: | Metrics:         1280 (mtu) 4 (length)
INFO: | Sharability:     0
INFO: | Main-Flow:       {{21.199.102.49 35392 } {10.7.0.9 27041 }}
INFO: | Hops: [1-ff00:0:112 1>2 1-ff00:0:110 1>41 1-ff00:0:111] MTU: 1280, NextHop: 127.0.0.25:31020
INFO:
INFO: SubFlow AZIF - 3 connection successful established!
INFO: | TCP-Flow:        84.132.127.112:39557 <-> 10.7.0.9:27041
INFO: | Net-Flow:        127.0.0.1:3363 <-> 1-ff00:0:111,127.0.0.1:27041
INFO: | Metrics:         1280 (mtu) 4 (length)
INFO: | Sharability:     0
INFO: | Main-Flow:       {{21.199.102.49 35392 } {10.7.0.9 27041 }}
INFO: | Hops: [1-ff00:0:112 1>2 1-ff00:0:110 1>41 1-ff00:0:111] MTU: 1280, NextHop: 127.0.0.25:31020
```

Figure 4.1: Output generated by the Shila instance running on Host 1. Note that iPerf3 creates two TCP connections, one to send the control signals and one for the actual data exchange.

Following, the output of certain involved entities is shown. Figure 4.1 displays the output generated by the Shila instance running on Host 1. The shim layer creates a default number of three paths through the SCION network used for the data exchange between the two iPerf3 instances. Figure 4.2 shows the output generated by iPerf3 running on Host 1. Figure 4.3 visualizes the distribution of the data exchange among the three different flows.

```
Host 1 - Starting iPerf3..

Connecting to host 10.7.0.9, port 27041
[  4] local 21.199.102.49 port 35392 connected to 10.7.0.9 port 27041
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.01   sec  1.88 MBytes  15.6 Mbits/sec    1   14.1 KBytes
[  4]   1.01-2.01   sec  1.68 MBytes  14.1 Mbits/sec    0   14.1 KBytes
[  4]   2.01-3.01   sec  1.60 MBytes  13.4 Mbits/sec    0   14.1 KBytes
[  4]   3.01-4.01   sec  3.03 MBytes  25.4 Mbits/sec    0   14.1 KBytes
[  4]   4.01-5.01   sec  1.94 MBytes  16.3 Mbits/sec    0   14.1 KBytes
[  4]   5.01-6.01   sec  2.31 MBytes  19.4 Mbits/sec    0   14.1 KBytes
[  4]   6.01-7.00   sec  2.05 MBytes  17.3 Mbits/sec    0   14.1 KBytes
[  4]   7.00-8.01   sec  1.46 MBytes  12.2 Mbits/sec    0   14.1 KBytes
[  4]   8.01-9.01   sec  1.42 MBytes  11.9 Mbits/sec    0   14.1 KBytes
[  4]   9.01-10.01  sec  2.24 MBytes  18.8 Mbits/sec    0   14.1 KBytes
[  4]  10.01-11.00  sec  2.46 MBytes  20.8 Mbits/sec    0   14.1 KBytes
[  4]  11.00-12.01  sec  1.83 MBytes  15.3 Mbits/sec    0   14.1 KBytes
[  4]  12.01-13.01  sec  2.32 MBytes  19.4 Mbits/sec    0   14.1 KBytes
[  4]  13.01-14.01  sec  2.45 MBytes  20.6 Mbits/sec    0   14.1 KBytes
[  4]  14.01-15.00  sec  2.54 MBytes  21.4 Mbits/sec    0   14.1 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-15.00  sec  31.2 MBytes  17.5 Mbits/sec    1             sender
[  4]   0.00-15.00  sec  30.3 MBytes  17.0 Mbits/sec                  receiver

iperf Done.
scion@mptcp-over-scion-vm-dev:~/go/src/shila/testing/local$
```

Figure 4.2: Output generated by the iPerf3 client instance on Host 1. Note that the measured metrics are not representative since the test run was done on a local setup of SCION.

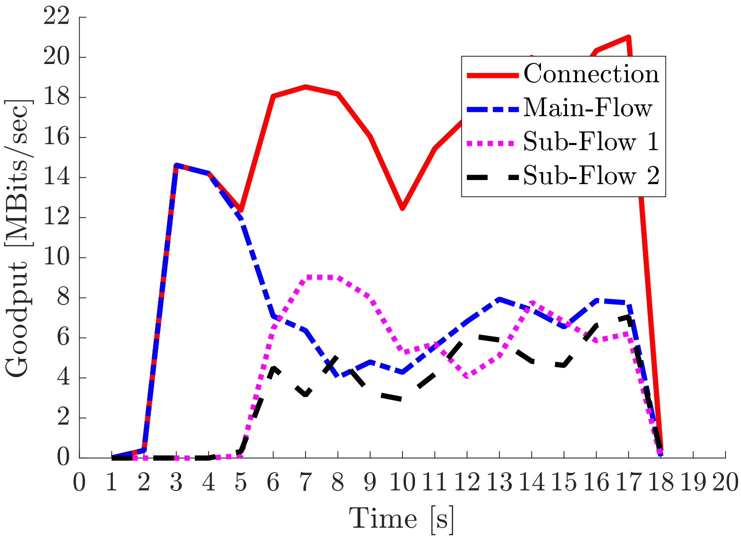

Figure 4.3: Plot showing how the achieved goodput for the connection between Host 1 and Host 2 is under hood distributed to the three flows created by Shila.

4.2 Functionality

The functionality of Shila can be divided into three parts. The first part considers the setup of Shila in general, i.e. everything necessary to get a running instance of Shila. The latter two parts specifically consider the data exchange between two TCP endpoints. In detail, the second part discusses what is necessary to establish such a connection and the third part covers the operating state and also the cleaning up once a connection is no longer needed.

In the following subsections the three parts are outlined, so the reader gets a basic understanding of the mechanics of Shila. In Chapter 5 everything mentioned is explained again in more detail.

Setup

Launching and setting up Shila includes the following steps.

Loading of the Routing-Information Shila reads the Routing-Information presented in the respective file. It contains the mapping from TCP addresses of applications to SCION addresses of hosts running these apps.

Setup of Namespaces Shila sets up two different network namespaces. The ingress namespace (*shila-ingress*) holds the network interface used to accept incoming connections. The egress namespace (*shila-egress*) accommodates the network interfaces which are used for connections initiated by the respective host.

Creation of virtual Interfaces A virtual network interface is a network device that is supported entirely in software, i.e. there is no real hardware network adapter. It allows applications to take over the role of the wire. For a real network interface, the IP datagrams of connected applications are sent directly to the cable. Virtual interfaces make these data available to applications in userspace.

The shim layer instantiates a single virtual network interface in the ingress namespace. This interface is per default bound to the IP address 10.7.0.9 and is used by all listening TCP applications to bind their sockets. In the egress namespace one to eight[2] virtual interfaces can be instantiated. Each is assigned a random IP address. Each of these virtual network interfaces later corresponds to a single flow of an MPTCP connection.

[2] The current implementation of MPTCP uses a maximum of eight different interfaces for a single MPTCP connection.

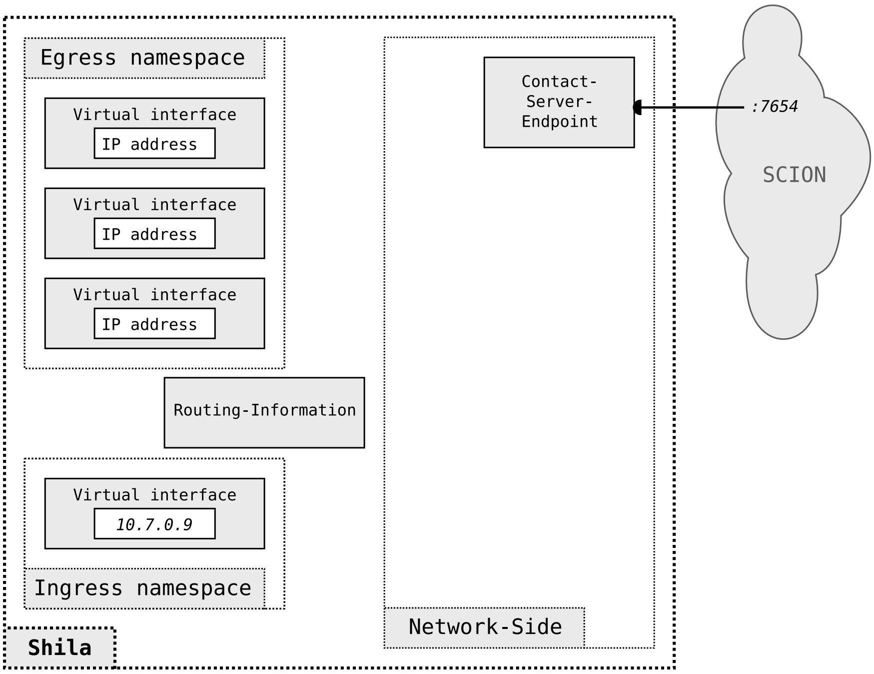

Figure 4.4: Illustration of the state of Shila once the setup is complete. Through the Contact-Server-Endpoint, Shila receives connection requests from other instances. Applications (server) can bind to the virtual interface in the ingress namespace. Requests from applications (client) are intercepted by Shila via the virtual interfaces in the egress namespace.

Running the Contact-Server-Endpoint To be receptive to incoming connections, Shila starts a so-called Contact-Server-Endpoint. It binds per default to the SCION port 7654 and serves as an initial point of contact. Shila instances running on different hosts initiate possible data exchange through this contact point.

Figure 4.4 illustrates the state of Shila once the setup is completed. The implementation is now ready to establish connections and provide data exchange between the client and server instance of TCP applications running on, possibly different, SCION hosts.

Connection Establishment

This subsection presents the individual steps of a connection establishment between a client instance of an application and its server counterpart over MPTCP on top of SCION. The discussion is based on the example of Subsection 4.1 and it assumes that a successfully initialized instance of Shila is running on Host 1 and Host 2. All the presented steps are illustrated in Figure 4.5.

The creation of a connection itself can be divided into two parts. The establishment of the initial TCP connection, also called Main-Flow, and the establishment of further TCP connections, the so-called Sub-Flows. For all upcoming parts of this thesis a connection always means the totality of all flows, i.e. the Main-Flow and all possible Sub-Flows. Next, a walk-through guides through the sequence of steps of a Main-Flow establishment.

Main-Flow Establishment

M-1 On Host 2, the server instance of iPerf3 starts listening for incoming TCP connections. It therefore binds to the virtual interface in the Shila ingress namespace on the TCP address 10.7.0.9:27041. This step is not recognized by Shila.

M-2 On Host 1, the client instance of iPerf3 tries to connect to its server counterpart on TCP address 10.7.0.9:27041. For the connection establishment an IP datagram, containing the SYN, is sent via one of the virtual interfaces in the egress namespace where it is intercepted by Shila.

M-3 Shila extracts the destination TCP address from this datagram and looks up the corresponding SCION address in the Routing-Information. For the example, this is 2-ff00:0:220,127.0.0.1, the SCION address of Host 2.

M-4 A Contact-Client-Endpoint is created. It establishes a backbone connection, called Contact-Backbone-Connection, over SCION to the Contact-Server-Endpoint listening on Host 2, SCION port 7654.

M-5 The SYN is forwarded to the server along the Contact-Backbone-Connection.

M-6 The Shila instance on the server-side hands over the received IP datagram, holding the SYN, to the virtual network interface in the ingress namespace. From there the datagram finds its destination, the iPerf3 server instance.

M-7 On Host 2, a Traffic-Server-Endpoint, listening on SCION port 27041, is created, ready to receive backbone connections.

M-8 On Host 1, a Traffic-Client-Endpoint is created. It establishes a so-called Traffic-Backbone-Connection to its counterpart on the server-side.

M-9 The IP datagram containing the answer from the iPerf3 server instance, a SYN+ACK, arrives at the Shila instance on Host 2. It is forwarded to the corresponding Traffic-Server-Endpoint. From there it is sent over the Traffic-Backbone-Connection to Host 1.

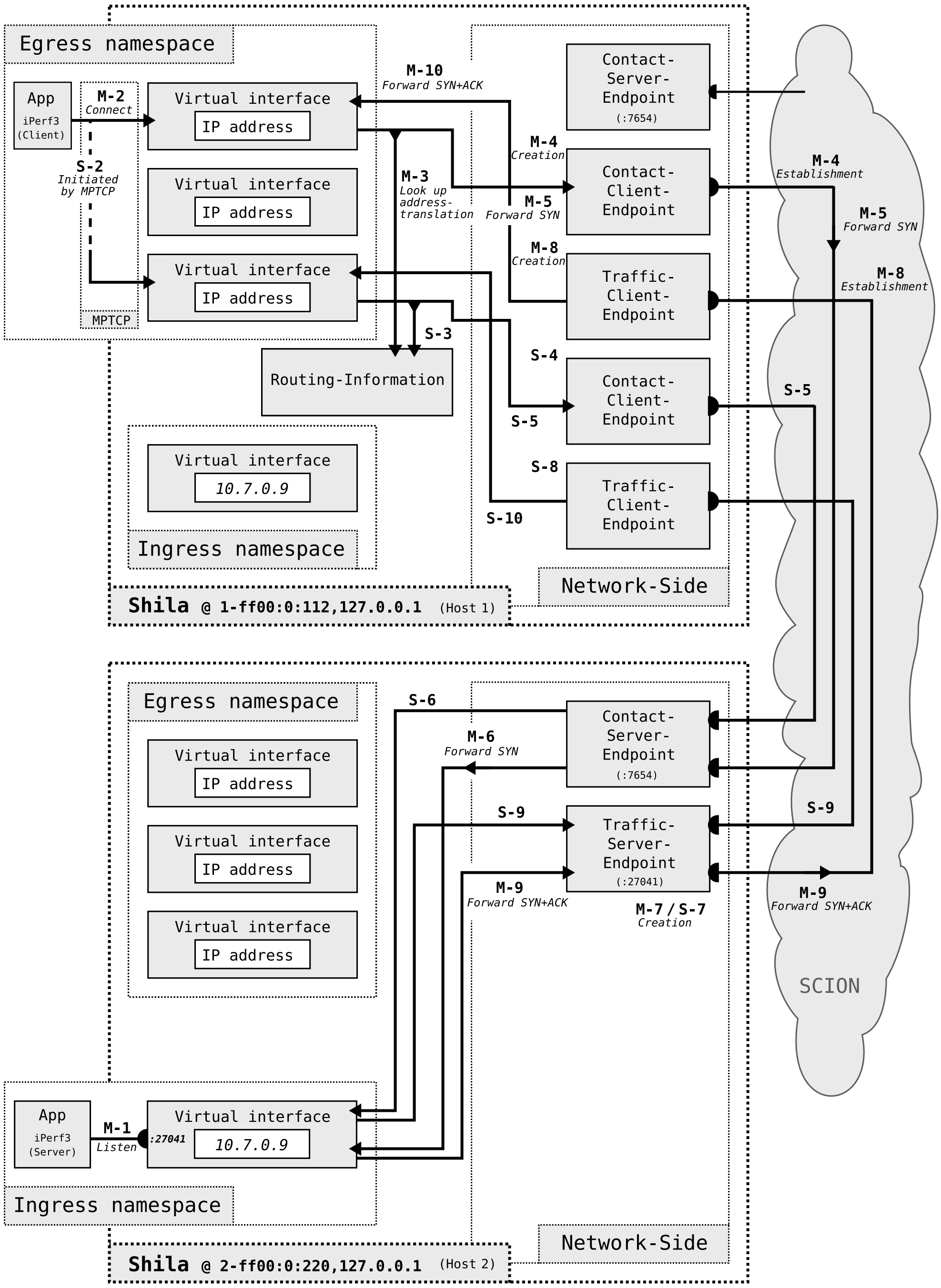

Figure 4.5: Illustration of the steps involved in the connection establishment of a Main-Flow (**M-1** to **M-10**) and a Sub-Flow (**S-1** to **S-10**).

M-10 The reception of the first data received from the Traffic-Client-Endpoint on Host 1 concludes the establishment of the Main-Flow. Shila extracts its identifier, denoted as Main-Flow-Identifier, and links it to the corresponding routing information. This mapping is important for the following creation of Sub-Flows. The IP datagram, holding the SYN+ACK, is finally handed to the virtual interface from where the initial SYN came from. From there it finds its way to the client instance of iPerf3.

As soon as the Main-Flow of a connection is established, MPTCP starts to initiate Sub-Flows. The procedure differs only in some points from the procedure for establishing a Main-Flow. These points are mentioned below. For illustration purposes, an established Sub-Flow is indicated in Figure 4.5.

Sub-Flow Establishment

(S-1) This step is not necessary, the server-side of the app is still listening.

S-2 The initial connection request for a Sub-Flow is not made by the client instance of the application but by MPTCP itself.

S-3 For a Sub-Flow, Shila extracts the identifier for the corresponding Main-Flow, i.e. the Main-Flow-Identifier, which is carried in the SYN datagram. It uses this identifier to get all the necessary information about the connection, e.g. the SCION destination address, from the Routing-Information.

S-7 For a Sub-Flow it is not necessary to create a new Traffic-Server-Endpoint which listens on SCION port 27041. Such an instance was already created during the establishment of the corresponding Main-Flow.

Once established, a flow is immediately used for the data exchange of the application. If at one point the connection is no longer needed, it is removed again, as mentioned in the following subsection.

Normal Operation and Clean-up

As soon as the Main-Flow or a Sub-Flow is established, its Contact-Backbone-Connection is no longer needed and is terminated. For data exchange between the TCP endpoints, data can be forwarded through all established flows. The decision along which flow the data is sent, is made by MPTCP and cannot be influenced by Shila[3]. Data forwarded along a certain flow travels via the respective endpoints along the Traffic-Backbone-Connection through the SCION network. If a connection is no longer needed, i.e. no more data is flowing through, the respective Traffic-Backbone-Connections are removed together with its endpoints.

[3]But of course, could Shila decided to send a flow through the Backbone-Connection of a different flow.

Chapter 5

Shila - Implementation

In this chapter, we present a more detailed treatise of the implementation of Shila. It builds on the explanations given in Chapter 4. First, an introduction of important terms and concepts is given followed by the description of the underlying implementation structure. Then we treat the three parts of functionality (setup, connection establishment and the normal operation) for a second time, but more detailed than in Chapter 4 and with reference to the structure.

5.1 Core Concepts and Terms

In the subsections below the concepts and terms important to understand the content of this chapter are explained. Figure 5.1 summarizes them in an illustration.

Flows

The implementation of Shila knows two types of so-called flows; the TCP-Flow and the Net-Flow. These flows are used to uniquely identify connections and enable internal mapping tasks.

TCP-Flow

The TCP-Flow consists of two TCP addresses representing the connection between two TCP endpoints. One of the two addresses is identified to be the source, the other to be the destination of the TCP-Flow. If used to identify a connection within an endpoint itself, then this endpoint is always the source of the TCP-Flow. Within data that is exchanged from one endpoint (source) to another (destination), the assignment is accordingly.

Net-Flow

The Net-Flow consists of two SCION addresses that represent the connection between two SCION endpoints. The same principles here as for the TCP-Flow presented above apply. In addition to the source and destination, the Net-Flow also contains the SCION path of the corresponding connection.

Shila-Endpoint

A Shila-Endpoint is a point of contact between Shila and the outside world through which data is received or sent. The different types of these endpoints will be presented in the course of this chapter. To avoid confusion with the naming, the hierarchy of the individual endpoints is listed below.

```
Shila-Endpoint
├─ Kernel-Endpoint
└─ Network-Endpoint
   ├─ Client-Endpoint
   │  ├─ Contact-Client-Endpoint
   │  └─ Traffic-Client-Endpoint
   └─ Server-Endpoint
      ├─ Contact-Server-Endpoint
      └─ Traffic-Server-Endpoint
```

Shila-Packet

For the exchange of data within Shila, so-called Shila-Packets are used. A packet contains the actual payload, namely the IP datagram exchanged between two TCP endpoints. It furthermore holds a link to its creating Shila-Endpoint and is associated with a TCP-Flow as well as Net-Flow.

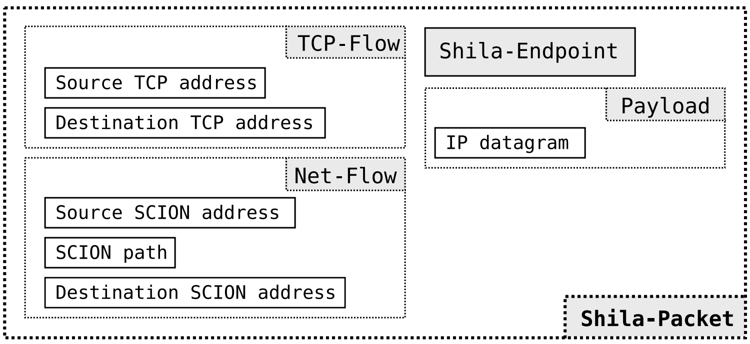

Figure 5.1: Illustration of the Shila-Packet containing the payload, a handle to its creating Shila-Endpoint and the associated TCP-Flow and Net-Flow.

5.2 Structure

In the outermost level, the implementation of Shila can be divided into five different parts, illustrated in Figure 5.2. The Kernel-Side handles the creation of and the interaction with the virtual interfaces and handles the data exchange with the TCP applications. The Network-Side is responsible for the creation and handling of backbone connections over SCION and the data exchange with Shila instances on other devices. The Working-Side in between receives incoming data from both sides and makes sure that it is handed over to the correct Shila-Connection. The Shila-Connection itself maintains the state of a connection and delivers the data to the desired destination. To be able to do this, the Shila-Connection relies on the services of the Router. The data exchange between the individual parts takes place via traffic channels, which transport the Shila-Packets. In the following, we discuss each of the parts in more detail and provide a more detailed subdivision of the structure into further subparts and their functionality.

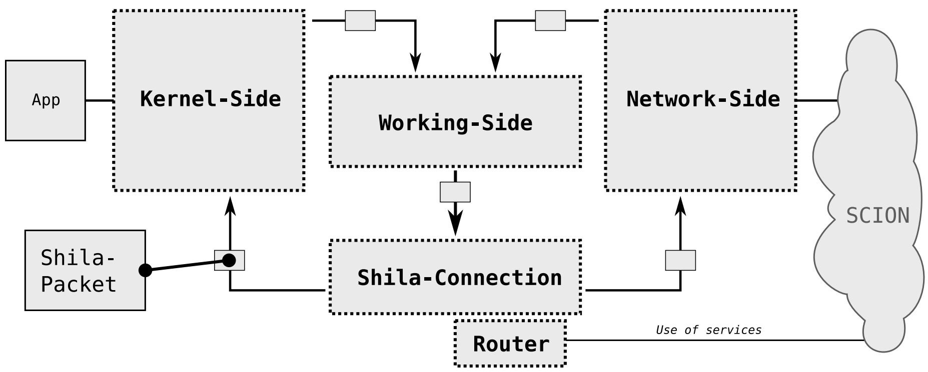

Figure 5.2: Illustration of the outermost division of the implementation into different parts. The Kernel-Side is responsible for the data exchange with the applications, whereas the Network-Side is responsible for the data exchange with other Shila instances. The Working-Side distributes each Shila-Packet to the correct Shila-Connection from where the packets are forwarded to their intended destination. The Router provides the necessary support to the Shila-Connection. To be able to do that, it uses services provided by the SCION network.

Kernel-Side

The Kernel-Side, shown in Figure 5.3, is managed by an appropriate manager, allocating the network namespaces and creating the individual Kernel-Endpoints, which are located within the namespaces. It furthermore pro-

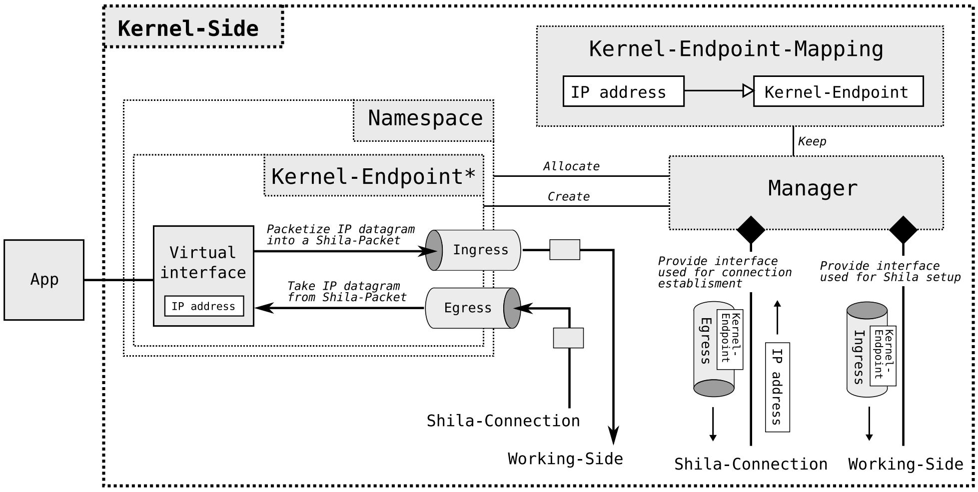

Figure 5.3: Illustration of the Kernel-Side with its subparts. The manager allocates the namespaces and creates Kernel-Endpoints in them. It keeps a mapping from the IP addresses to the created Kernel-Endpoints. Every Kernel-Endpoint contains a virtual interface with an IP address assigned to it and holds two traffic channels. One for sending incoming data to the Working-Side and one to receive outgoing data from the Shila-Connections. Entities can request handles to these traffic channels via the interface provided by the manager in the Kernel-Side.

vides an interface for the creation of Kernel-Endpoints and it allows to query for the traffic channels held by these endpoints. The Kernel-Endpoint-Mapping stores a mapping from IP addresses to the corresponding Kernel-Endpoints.

Kernel-Endpoint

The Kernel-Endpoint creates and holds the virtual network interface over which the data exchange with the TCP endpoints takes place. Every Kernel-Endpoint is identified by an IP address, namely the IP address assigned to its underlying virtual interface. It also holds two traffic channels. For an IP frame received through the virtual interface, the TCP-Flow is determined and a newly created Shila-Packet is sent via the ingress channel to the Working-Side. Through the egress channel packets from Shila-Connections are received. From these packets, the payload is taken and transferred to the virtual interface and from there to the TCP endpoint.

Virtual Interface

Shila uses the universal TUN/TAP device driver [25] to set up virtual network devices on the host. Such a virtual network device is perceived by the kernel as a real Ethernet device. The difference is that traffic from the kernel is not sent through a physical medium but can be read by an application, in our case Shila. Furthermore receives the virtual device the data not over the cable but from an application, again Shila.

Each virtual network interface has an IP address assigned to it. When a TCP endpoint sends data over a virtual device, Shila receives the corresponding IP datagrams. Each of these contains a TCP-Flow. The source TCP address consists of the IP address of the virtual device and an arbitrary port. The destination address is the TCP address the application wants to send data to. Conversely; when Shila writes payload (i.e. an IP datagram) to a virtual device, then the kernel forwards it to the application identified by the destination TCP address.

Network-Side

The Network-Side is also orchestrated by an appropriate manager. At startup, the manager takes care of establishing the Contact-Server-Endpoint. It furthermore provides the interface to Shila-Connections for the creation of new Contact-Client-Endpoints, Traffic-Client-Endpoints and Traffic-Server-Endpoints. The Network-Side holds three different mappings. Two to get the corresponding Client-Endpoint for a given TCP-Flow, and one to get the Server-Endpoint associated with the source SCION address. These mappings are only used by the manager to interact with the endpoints, e.g. for the deallocation in case of a disconnection. They are not needed for the data exchange itself. After establishing the connection, the corresponding Shila-Connections directly hold a handle of the traffic channels from the respective endpoints. Figure 5.4 illustrates these Network-Endpoint-Mappings as part of the whole Kernel-Side and its subparts.

Server-Endpoint

The Server-Endpoint is one of two types of Shila-Endpoints found on the Network-Side. It can be further divided into Contact-Server-Endpoint and Traffic-Server-Endpoint. However, there is no difference in basic functionality between the two. We mention the differences in the discussion of setup and connection establishment, which can be found in Section 5.3 and Section 5.4.

A Server-Endpoint holds and handles a collection of Backbone-Connections, i.e. connections through the SCION network to Client-Endpoints running on other instances of Shila. Each such Backbone-Connection is associated with a Net-Flow. The Server-Endpoint-Mapping[1] in the Network-Side uses the source address of this Net-Flow as a key. Inside the endpoint, the corresponding destination address is used as the key for the Backbone-Connection-Mapping.

A Backbone-Connection itself uses the appnet package [31] to establish the connection in SCION. Data received from the SCION network is packed into a Shila-Packet by the responsible Backbone-Connection and sent via the ingress traffic channel of the server to the Working-Side. Packets received from Shila-Connections are allocated to their Backbone-Connection using their Net-Flow and the Backbone-Connection-Mapping. From there, the payload is put into a Shila-Message and sent via appnet package through the SCION network.

Client-Endpoint

The Client-Endpoint is the other type of Shila-Endpoints within the Network-Side. It is the initiating party of a Backbone-Connection and is associated with just a single Net-Flow. As with the Server-Endpoint, a distinction is made between a Contact-Client-Endpoint and a Traffic-Client-Endpoint, which again is discussed later. The data exchange between the SCION network and the internal Shila units works via the appnet package and two traffic channels. Since only one Backbone-Connection is assigned to a single Client-Endpoint, no mapping is required inside.

Working-Side

Upon the arrival of a Shila-Packet at the Working-Side it is processed by a dedicated worker. Its TCP-Flow is extracted and used as the key for the Shila-Connection-Mapping, the heart of the Working-Side. It contains the mapping from TCP-Flows to Shila-Connections. If there is no corresponding entry available then a new Shila-Connection is created and inserted into the mapping. Once obtained, the responsible worker processes the Shila-Packet through the associated Shila-Connection. The described process is illustrated in Figure 5.5.

Shila-Connection

The Shila-Connection, depicted in Figure 5.6, is the main part of Shila, with its logic responsible for establishing and maintaining connections. The centerpiece is the state machine and the close coupling to the Router, which provides SCION destination and path information. During connection establishment, the Shila-Connection uses the interfaces of Kernel-Side and

[1]In Figure 5.4, this mapping is shown as part of the Network-Endpoint-Mappings.

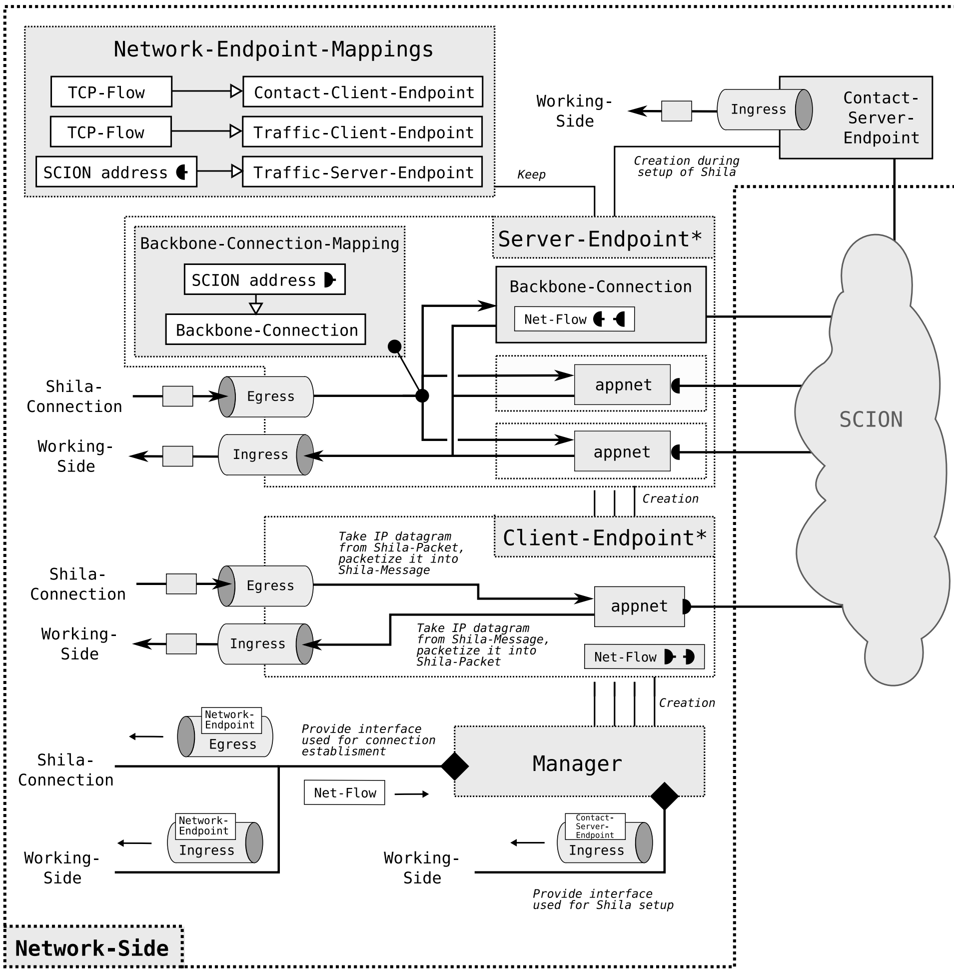

Figure 5.4: Illustration of the Network-Side with its subparts. The manager is responsible for the creation of the Contact-Server-Endpoint during the setup of Shila. It furthermore provides an interface for the creation of new Network-Endpoints required for the establishment of new connections. The handles to the respective traffic channels are returned once the endpoints are ready. Every Backbone-Connection is initiated by a Client-Endpoint contacting a listening Server-Endpoint. To be able to send the outgoing traffic through the correct Backbone-Connection, every Server-Endpoint contains a Backbone-Connection-Mapping. This mapping is the only mapping on the Network-Side which is invoked for every packet sent out via the Network-Side. The Network-Endpoint-Mappings are only used by the manager for maintenance.

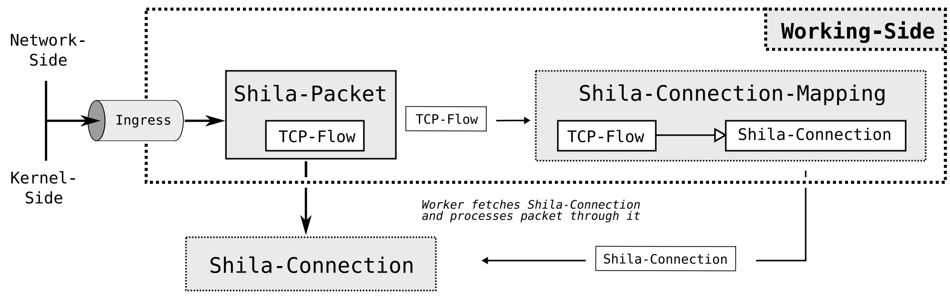

Figure 5.5: Illustration of the Working-Side. Every Shila-Packet arriving at the Working-Side gets processed by a worker. This worker uses the TCP-Flow of the packet as key into the Shila-Connection-Mapping. If for a given Shila-Packet and its TCP-Flow no connection exists yet, a new one is created. The worker then processes the Shila-Packet through the Shila-Connection. Note that the establishment of the connection itself happens inside of the Shila-Connection.

Network-Side as well as the functionalities provided by the Router to put its traffic channels into place. Once established, the packets flow smoothly through the Shila-Connection to their respective destination endpoints without much interaction. Every Shila-Connection is identified by its TCP-Flow and holds, once the connection is established, the corresponding Net-Flow.

Router

The Router supplies the Shila-Connection with the Net-Flow as part of the connection establishment. It is connected to the SCION daemon running on the host and has access to a hardcoded mapping between TCP and SCION addresses. This mapping is denoted as Routing-Information. The Router illustrated as part of Figure 5.6 also maintains a Routing-Mapping which uses TCP-Flow as a key to store Shila-Routing-Entries and an additional mapping from Main-Flow-Identifier to TCP-Flows.

Shila-Routing-Entry

A Shila-Routing-Entry stores a SCION address and several paths leading through the SCION network to this destination. The number of stored paths depends on the setting of Shila, namely on the number of virtual interfaces created in the egress namespace. Also stored for each of the paths is its quality, given in different metrics. These metrics are used as a decision criterion in the process of path selection. This is itself part of the creation of routing entries and is described in its own Section 5.5.

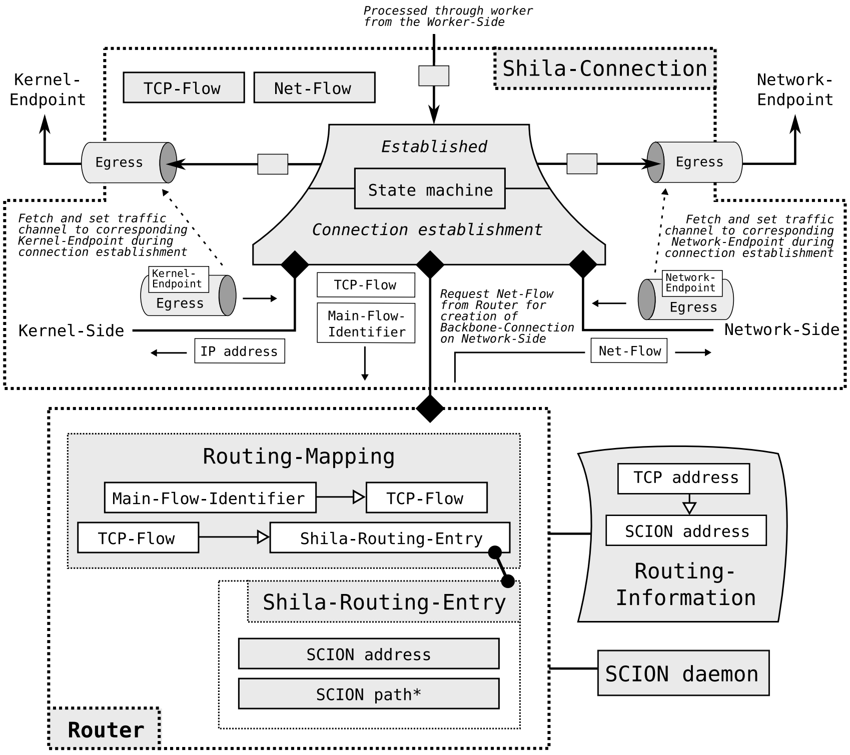

Figure 5.6: Illustration of the Shila-Connection together with the Router. The Shila-Connection receives the packets assigned to it through the Working-Side. If the connection is already established, the packet is forwarded directly to its destination via the egress traffic channel of the corresponding Shila-Endpoint. Coming from the Kernel-Side a packet leaves via the channel of a Network-Endpoint, coming from the Network-Side it leaves via the channel of a Kernel-Endpoint. During the connection setup, the Shila-Connection first uses the Router to determine the Net-Flow. It then requests the corresponding egress traffic channel from the Network-Side using this Net-Flow. This request involves the creation of the Backbone-Connection and the associated Network-Endpoints. By using the source IP address of the TCP-Flow, the Shila-Connection gets the appropriate egress traffic channel from the Kernel-Side. The connection is established once both traffic channels are in place.

5.3 Setup

In this section, we describe the crucial steps executed during the setup of Shila. The description is similar to the one given in Section 4.2. This time its more detailed and also refers to the structure presented in the sections and illustrations above.

Loading of the Routing-Information To perform its function of providing translations between a TCP-Flow and a Net-Flow, the Router loads the Routing-Information from disk. Saved as a JSON file it contains the translation from a TCP to a SCION address. Currently, there is no other way to provide the Router with this necessary information. However, alternatives are certainly conceivable and are mentioned in Chapter 7. To be able to do path selection the router furthermore connects to the SCION daemon.

Setup of Namespaces In a first step, the manager on the Kernel-Side creates the two network namespaces [43, 28], the ingress and the egress namespace. System resources, like network devices or IP routing tables, associated with networking and assigned to different namespaces are isolated from each other. This simplifies the development and setup of the Kernel-Side but on the downside requires the TCP applications to be started within the corresponding namespaces. For further discussions with regard to namespaces see Chapter 7.

Creation of Kernel-Endpoints hosting virtual Interfaces In the second step, the manager creates the Kernel-Endpoints in the appropriate namespaces. In the ingress namespace, this is exactly one, in the egress namespace, this is a number between one and a maximum of eight. The number of Kernel-Endpoints selected in the egress namespace later corresponds to the number of paths used for a connection. The current implementation of MPTCP supports up to eight interfaces. Implementing Shila to support more is therefore useless, but can be realized without problems if necessary in the future.

Each Kernel-Endpoint contains a virtual interface to which an IP address is assigned. This address is also assigned to the Kernel-Endpoint itself and is used as a key in the Kernel-Endpoint-Mapping. The virtual interface in the ingress namespace is assigned a default address (10.7.0.9), the assignment in the egress namespace is done at random. Upon its establishment, every Kernel-Endpoint creates two traffic channels. The ingress channel, which is used to forward incoming data, is registered at the Working-Side.

Contact-Server-Endpoint During the setup, the manager in the Network-Side creates just one Network-Endpoint; the Contact-Server-Endpoint. This endpoint starts listening for incoming Backbone-Connections on the default

SCION port 7654 using the appnet package. In contrast to all other Shila-Endpoints, it only needs the ingress traffic channel, namely for incoming messages that initiate the establishment of a new connection. This channel is registered with the Working-Side.

After the successful setup, applications can be started in the respective namespaces and the data exchange is initiated with the connection establishment.

5.4 Connection Establishment

In this section, we again discuss the individual steps in the establishment of a new connection between two TCP endpoints. We use the explanation of Section 4.2 as a basis, but go into more detail and extend the sequence with intermediate steps, referring to the treatise of the structure given in Section 5.2. The starting point and the example used is the same one as used in the introductory treatise of the connection establishment in Section 4.2. Illustration 5.7 shows the state after the successful establishment of the connection with the values from the example discussed. These values are also mentioned in the discussion of the individual steps below. In the used notation, the source comes first followed by the destination.

Main-Flow Establishment

We start with a more detailed discussion of the Main-Flow establishment.

M-1 On Host 2, the server instance of iPerf3 starts listening for incoming TCP connections. It, therefore, binds to the virtual interface in the Shila ingress namespace on the TCP address 10.7.0.9:27041. This step is not recognized by the Kernel-Endpoint hosting the corresponding virtual interface.

M-2 On Host 1, the client instance of iPerf3 tries to connect to its server counterpart on TCP address 10.7.0.9:27041. For the connection establishment, an IP datagram, containing the SYN, is sent via one of the virtual interfaces in the egress namespace. Let us say it is identified by the IP address 10.0.0.1 and the socket bound to port 11111.

M-3 The Kernel-Endpoint owning the corresponding virtual interface receives the IP datagram and puts it into a Shila-Packet. While payload, Shila-Endpoint and TCP-Flow[1] are determined, the Net-Flow for the created Shila-Packet is not yet known. The packet is now forwarded to the Working-Side.

[10.0.0.1:11111, 10.7.0.9:27041][1]

M-4 The Working-Side uses the TCP-Flow, extracted from the packet, as the key into the Shila-Connection-Mapping. Since there is no corresponding Shila-Connection, a new one is created and the Shila-Packet is forwarded on to it.

M-5 To be able to forward the received Shila-Packet, the newly created Shila-Connection has to first determine the destination SCION address and path. It does this by querying the Router, which determines the corresponding parts of the Net-Flow for the TCP-Flow of the packet.

The entire process of determining the Net-Flow, i.e. address translation and the path selection, is explained in the following Section 5.5. To include the explanation as part of this sequence, would make it unnecessary bulky to read. For the next step M-6, we assume that the required parts of the Net-Flow are determined.

M-6 The Shila-Connection uses the interface of the manager on the Network-Side to initiate the creation of a Contact-Client-Endpoint. It provides the Net-Flow[2] it has received from the Router.

[1-ff00:0:112,127.0.0.1, 2-ff00:0:220,127.0.0.1:27041][2]

M-7 The Contact-Client-Endpoint establishes the so-called Contact-Backbone-Connection to the Contact-Server-Endpoint. It uses the source address available in the provided Net-Flow together with the default SCION port 7654 of the listening server. For our example, we assume that the outgoing SCION port of the Contact-Client-Endpoint is 12121. This information completes the Net-Flow of the Contact-Backbone-Connection[3].

[1-ff00:0:112,127.0.0.1:12121, 2-ff00:0:220,127.0.0.1:7654][3]

M-8 With the successful setup of the Contact-Client-Endpoint, the Shila-Connection receives the handle to its egress traffic channel. Using this channel, the Shila-Packet, and hence the initial IP datagram containing the SYN is now transferred to the Contact-Server-Endpoint running on the Shila instance on Host 2.

M-9 The IP datagram is received at the Contact-Server-Endpoint on Host 2. There, it is put into a Shila-Packet and forwarded via the ingress traffic channel to the Working-Side. The Net-Flow[4] (namely that of the backbone connection between Contact-Client-Endpoint and Contact-Server-Endpoint) can already be assigned to the newly created Shila-Packet. The TCP-Flow[5] is also known since it was transmitted with the establishment of the Contact-Backbone-Connection.

[2-ff00:0:220,127.0.0.1:7654, 1-ff00:0:112,127.0.0.1:12121][4]
[10.7.0.9:27041, 10.0.0.1:11111][5]

M-10 The Working-Side on Host 2 cannot find a Shila-Connection for the TCP-Flow[5] in the Shila-Packet. Accordingly, a new one is created and the packet is passed to it.

M-11 Inside the Shila-Connection, three tasks are performed. First, the Net-Flow[4] of the Shila-Packet is extracted and assigned to the Shila-Connection. Second, the Shila-Packet is forwarded. The source IP address of its TCP-Flow[5] is used to retrieve the egress traffic channel from the Kernel-Endpoint residing in the ingress namespace. This is done by the Shila-Connection by querying the Kernel-Endpoint-Mapping inside the Kernel-Side. For our example, the IP address used as the key is 10.7.0.9. The Shila-Packet is forwarded through the respective channel, once it is obtained. Third, the Shila-Connection causes the creation of a Traffic-Server-Endpoint which is ready for new Backbone-Connections at SCION port 27041. The ingress traffic channel is registered with the Working-Side and the Shila-Connection receives a handle to the respective egress traffic channel.

M-12 The Kernel-Endpoint receives the Shila-Packet, extracts its payload and sends the IP datagram containing the SYN through the virtual interface towards the listening iPerf3 server instance.

M-13 In the meantime, the Shila-Connection of Host 1 has initiated the creation of a Traffic-Client-Endpoint. This now establishes a Backbone-Connection to its opposite, called a Traffic-Backbone-Connection. This time, in comparison to step M-7, using the complete Net-Flow. Let us assume that the Traffic-Client-Endpoint bound locally to SCION port 34343, completing the Net-Flow[6] of the Traffic-Backbone-Connection. This Net-Flow is assigned to Shila-Connection on Host 1.

[1-ff00:0:112,127.0.0.1:34343, 2-ff00:0:220,127.0.0.1:27041][6]

M-14 With the successful establishment of the Traffic-Backbone-Connection, the involved Server-Endpoint on Host 2 receives the TCP-Flow[5] as well as two Net-Flows. The Net-Flow[4] of Contact-Backbone-Connection and the Net-Flow[7] of the Traffic-Backbone-Connection. Both of its destination addresses are used as key to store the Traffic-Backbone-Connection in the Backbone-Connection-Mapping. This is required since the Shila-Connection sends all data back, including the data received via the Contact-Server-Endpoint, via the Traffic-Server-Endpoint. Until now, the Shila-Connection on Host 2 held the Net-Flow of the Contact-Backbone-Connection. It can now can be updated to the one of the Traffic-Backbone-Connection.

[2-ff00:0:220,127.0.0.1:27041, 1-ff00:0:112,127.0.0.1:34343][7]

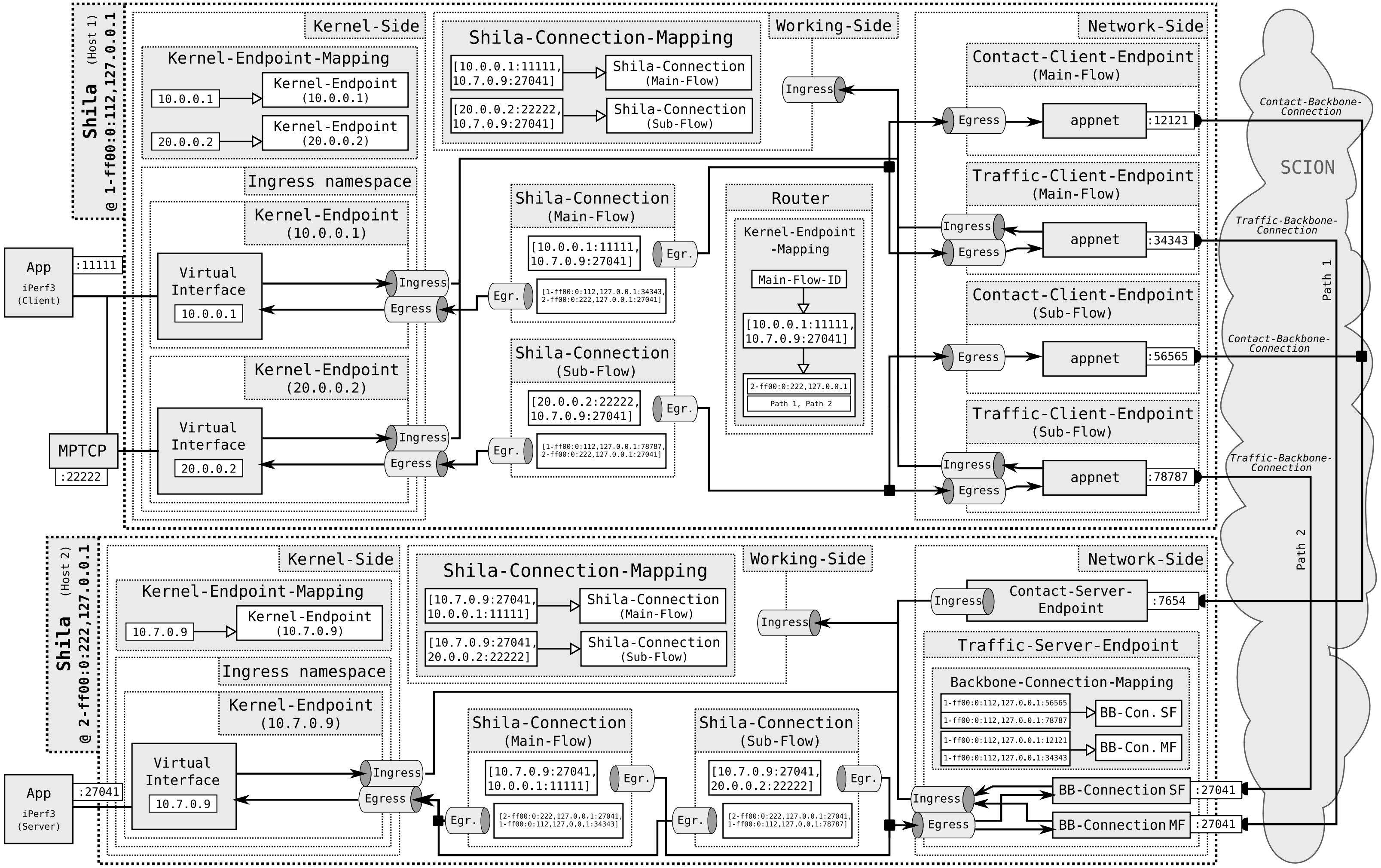

Figure 5.7: Illustration of the state of the Shila instances after the establishment of the Main-Flow and one Sub-Flow. Note that the Contact-Backbone-Connections are removed as soon as the establishment was successful. We show them just for demonstrative purpose.

M-15 The IP datagram containing the answer from the iPerf3 server instance, a SYN+ACK, arrives at the Kernel-Endpoint at Host 2. From there the answer travels in a Shila-Packet, using its TCP-Flow[5], through the Working-Side to the corresponding Shila-Connection.

M-16 The Shila-Connection assigns its Net-Flow to the Shila-Packet and forwards it via traffic channel to the corresponding Traffic-Server-Endpoint. From there the SYN+ACK finds its way through the Traffic-Backbone-Connection to the corresponding client part. The right connection is found by using the destination address of the packets Net-Flow as key into the Backbone-Connection-Mapping.

M-17 The reception of the first data from the Traffic-Client-Endpoint on Host 1 concludes the connection establishment. The first response from the iPerf3 server instance reaches the Shila-Connection, where the Main-Flow-Identifier is extracted. This identifier is used to update the Router accordingly such that Sub-Flows created later can be assigned to their Main-Flow. The response, holding the SYN+ACK, is then sent along the traffic channel toward the client instance of iPerf3.

M-18 The Contact-Backbone-Connection is no longer needed and can therefore be removed, together with its corresponding Network-Endpoints.

As soon as the Main-Flow is established, MPTCP starts to initiate further flows, triggering the establishment of Sub-Flows over the other virtual interfaces.

Sub-Flow Establishment

The establishment of Sub-Flow works in the same way as the one of the Main-Flow. Therefore, in the following, only the deviating points are mentioned. Of course, the individual entities bind to other ports. This is not explicitly mentioned and should be clear from Figure 5.7.

S-2 The initiation for the Sub-Flow is made by MPTCP itself. It binds to one of the yet unused virtual interfaces on Host 1. For our example we assume that port 22222 is chosen.

S-5 The Router already has an entry, holding destination address and paths. For the establishment of the Sub-Flow, it is sufficient for the Shila-Connection to submit the Main-Flow-Identifier to get the required Net-Flow.

5.5 Path Selection

In this section, we discuss the process of determining the Net-Flow given a TCP-Flow or Main-Flow-Identifier. It takes place in the Router and is illustrated as part of Figure 5.6.

During the connection establishment, the Router gets a request from the involved Shila-Connection. It provides either a TCP-Flow if a Main-Flow is about to be established or a Main-Flow-Identifier if a Sub-Flow is about to be established. We start with the explanation of the former followed by the latter.

Path Selection in the Main-Flow Establishment

The Shila-Connection is requesting the Net-Flow for a given TCP-Flow. That is, the Router has to determine and handle the three different entries.

Source SCION address This is the address of the host running the current Shila instance and can be requested from the SCION daemon.

Destination SCION address The Router first extracts the destination TCP address from the TCP-Flow. Then a request, using the extracted address as key, is placed into the Routing-Information to fetch the corresponding destination SCION address.

SCION path By querying the SCION daemon the Router receives a set of possible paths leading to the previously fetched destination address. From this collection, the Router selects the best possible subset and stores it together with the destination address in a Shila-Routing-Entry. The Routing-Mapping is accordingly updated with the TCP-Flow and the Main-Flow-Identifier such that the entry is later found when needed in the routing for a Sub-Flow establishment. Finally, the best path from the extracted subset is selected for the Net-Flow.

It remains to be clarified by which criteria the Router determines the best subset of paths. Its cardinality corresponds to the number of virtual egress interfaces. For a single connection, each interface corresponds to a possible flow for which a path is required. As quality criteria, three different metrics are available: Maximum Transmission Unit (MTU), path length and Sharability [50]. Sharability counts how many times each path segment occurs in a set of paths. It is a measure of the path distinctness. Which metric to use, is decided by the user when starting up Shila. The first two metrics, MTU and path length, are provided directly by the SCION daemon, the Router just has to sort the selection of paths accordingly and select the topmost ones. For MTU, the sorting is done in descending order, whereas for the path length in ascending order. For Sharability, the Router has to determine the minimizing subset which is computationally expensive. We, therefore, approximate the solution with a greedy approach.

Path Selection in the Sub-Flow Establishment

The Router gets a Main-Flow-Identifier in the case where the Shila-Connection is about to establish a Sub-Flow. In this case, a Main-Flow has already been created and a corresponding entry exists in the Routing-Mapping. The Router receives the corresponding TCP-Flow through the Main-Flow-Identifier and thus also the Shila-Routing-Entry. The Net-Flow is compiled with ease, as all required entries are either easily determined (source address) or are already stored in the entry (destination address, path). For the path, the next best-unused one is taken from the subset of flows stored in the fetched routing entry.

5.6 Normal Operation and Clean-up

Once a connection has been established, all the data exchanged between two TCP endpoints flows through the Shila-Connections and Backbone-Traffic-Connections of the involved flows. For each IP datagram traveling with MPTCP over SCION, there are either two or three lookups within the participating Shila instances required. The lookup in the Shila-Connection-Mapping is inevitable on both endpoints. If the datagram travels from server to client an additional one in the Backbone-Connection-Mapping is necessary. Also always necessary is the parsing of the IP datagram after its retrieval from the virtual interface, required for the determination of the TCP-Flow. Other than that, the processed Shila-Packet can pass the Shila instances without much interaction. Every Shila-Connection is able to relay the packets directly between the Shila-Endpoints using the handles of the respective traffic channels.

As mentioned in the introductory section, Shila has no influence on which flow of the connection the data uses. This is determined by MPTCP and depends among others on the choice of the congestion algorithm or the scheduler. It is generally not possible for Shila to interact with an established connection, it mainly acts as a relay station.

Shila maintains an established connection as long as there is data flowing. If there is no data exchange observable for a certain amount of time, the connection is terminated. This means that the Traffic-Backbone-Connections is closed and the associated infrastructure is removed. Optimally, the time out value for removing a connection is greater than the keepalive time [45] of the TCP connection.

5.7 Remark

In addition to a structure that is as modular as possible and a clear separation of the individual functionalities, the completion of a functional version of Shila had high priority within this work. Based on the first working version, measurements could be carried out to evaluate the performance of Shila for the first time. We discuss these performance measurements in the following Chapter 6. It should be considered, that a first implementation has prototype character in some sense, and that the quality and performance can be further improved in revision cycles. Hence, possible extensions and inputs for future work is discussed in Chapter 7.

Chapter 6

Performance Evaluation

This chapter contains the examination of Shila. Firstly, the questions to be answered are presented, followed by a description of the setup of the conducted measurements. Finally, the presentation of the results and conclusion is stated.

6.1 Questions of Interest

The evaluation presented in this chapter aims to find answers about the performance of Shila. The following questions are to be answered:

1. How does the performance behave in relation to the number of paths used for a connection?
2. How does the path selection influence performance?
3. How well does Shila compare to QUIC over SCION?

For the first two questions, we use the goodput[1] as a measure. For the third question, we additionally compare throughput. Upcoming, we present the setup used to gather these performance metrics.

6.2 Setup

Two different setups were used to gather the values required for answering the questions of interest. We denote the generation of measurements for MPTCP over SCION as Shila-Measurement. The measurement to obtain comparative values for QUIC over SCION we name Quic-Measurement. In the two upcoming subsections, the details about the setup, named accordingly, for these two recordings are presented.

[1] Goodput is the throughput on application-level, i.e. the amount of useful data exchanged per unit of time.

Setup Shila-Measurement

With the hereafter presented setup, we measure goodput and throughput between two hosts performing a data exchange. The involved hosts use iPerf3 as an application, relying on the functionality of Shila for data transfer via MPTCP over SCION.

All experiments of the measurement are run in the SCIONLab [33]. Three custom ASes, running on machines hosted by DigitalOcean [9], are connected to a subset of the available SCIONLab attachment points. These points are selected such that both, shorter inter-European and longer overseas connections, are represented. In Table 6.1 we list the mentioned custom ASes associated with their attachment point. Also included in the table are all other parameters of this measurement. A single experiment corresponds to the data exchange between two of the custom ASes using iPerf3 for a fixed number of paths and a certain path selection´. For Multipath TCP we use the default settings with which the functionality is installed. The value for goodput is directly extracted from iPerf3. For the throughput, we additionally log the incoming SCION data traffic on the receiving host using TShark [13] and calculate the throughput offline.

Data exchange is performed between all distinct pairs of the involved ASes in both directions and repeated ten times for the same set of parameters. The order of experiments is chosen randomly so that any fluctuations in the network are distributed over all realizations. After every experiment, the state of all involved entities is completely reset.

Custom ASes	mptcp-over-scion-as-0 connected to ETHZ-AP	
	mptcp-over-scion-as-1 connected to Magdeburg AP	
	mptcp-over-scion-as-2 connected to CMU AP	
TCP Application	iPerf v. 3.0.11	
	--time	30 s
	--set-mss	1024 Byte
Shila	Number of paths	1, 2, 4, 6, 8
	Path selection	MTU, Path length, Sharability
MPTCP	Stable release v0.95	
	Path manager	fullmesh
	Scheduler	default
	Congestion Control	CUBIC
Repetitions	10	

Table 6.1: Parameters of the setup used for the Shila-Measurement.

Setup Quic-Measurement

With the hereafter presented setup, we again measure goodput and throughput between two hosts doing a data exchange. This time a sending host uses a custom implementation to send a fixed amount of 50 MBytes using QUIC over SCION. By measuring the time it takes for the data transfer we are able calculate the goodput. The throughput is determined in the same way as with Shila-Measurement. Also identical is the network infrastructure and its topology. Every experiment, i.e. every data exchange between a pair of hosts is repeated ten times. The order of the experiments is chosen at random. The complete set of parameters is listed in Table 6.2.

Custom ASes	mptcp-over-scion-as-0 connected to ETHZ-AP	
	mptcp-over-scion-as-1 connected to Magdeburg AP	
	mptcp-over-scion-as-2 connected to CMU AP	
Application	Custom application	
	Data transferred	50 MByte
Repetitions	10	

Table 6.2: Parameters of the setup used for the Quic-Measurement.

Remark

The raw data generated during the measurements, the custom implementation for the Quic-Measurement as well as all the scripts to run and evaluate the measurements can be found in the Shila code repository [10].

6.3 Results

Influence of Path Count

Figure 6.1 shows the goodput obtained for the Shila-Measurement using the path length as the criterion for the path selection. Using multiple paths has a positive effect on the achieved performance. Using four instead of only one path increases the average good output by 20%. The large deviation of the average value is due to the different performances of the individual measured connections. The shorter inter-European connections achieve higher goodput than the longer overseas connection. The shape of the curves does not differ, however. The evaluation of the results obtained with the other two path selection algorithms resulted in equal values and behavior, observable in Table 6.3.

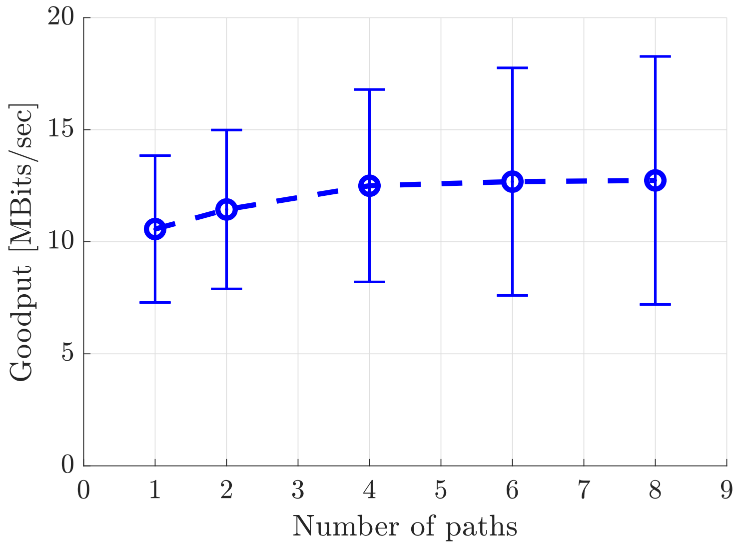

Figure 6.1: Plot of goodput depending on the number of paths used. An increasing number of paths leads to an increase in goodput. Path selection was done by minimizing the path length.

Influence of Path Selection

Table 6.3 shows the goodput obtained for the Shila-Measurement for all three path selection criteria. The chosen criterion does not influence the achieved performance. Reasons for this can be found in the topology of the SCIONLab network. Table 6.4 shows the percentage deviation to the optimal value of the other metrics for each path selection algorithm. The optimal value of a certain metric corresponds hereby to the value obtained by optimization with the corresponding path selection algorithm.

For all possible paths in the network, the same value for the MTU was obtained, irrespective of what metric for the selection was used. A possible positive effect of choosing paths with a higher MTU could therefore not be recognized at all.

Comparing the average length of the paths, obtained when optimized for minimal path length and maximal MTU, no deviation was found. A possible negative effect of longer paths when using MTU instead of path length could therefore also not be observed. If we now bring the Sharability into play, we notice that an optimization by MTU or path length leads to a deterioration of the Sharability value. However, this has no negative impact on the goodput achieved. Such a negative effect could especially then be observed, if bottlenecks in the network can be avoided by choosing the most different paths. If all shared links have sufficient capacity, this doesn't matter. We can

Paths	MTU	Path length	Sharability
1	10.37 ± 3.17	10.57 ± 3.28	10.34 ± 3.24
2	11.67 ± 3.72	11.44 ± 3.55	11.26 ± 4.18
4	12.55 ± 4.34	12.5 ± 4.29	12.09 ± 4.64
6	12.45 ± 5.32	12.68 ± 5.08	12.75 ± 5.37
8	12.63 ± 5.37	12.74 ± 5.53	12.46 ± 5.72

Table 6.3: Listing of the achieved goodput in MBits/sec for different numbers of paths and path selection criteria. The algorithm used for path selection does not influence performance.

therefore assume, that there were no capacity bottlenecks in the used links at the time the measurement was conducted.

Concerning the Sharability values in Table 6.4, two observations are noteworthy:

The values for the two path selection criteria MTU and Path length are almost identical. It is conceivable that the same selection resulted from both path selection procedures[2]. The path selection mechanism of Shila itself does not implement an explicit tiebreaker. However, the paths received from the SCION infrastructure may be already sorted by a criterion, in this case the length.

It is furthermore noticeable that the deterioration of Sharability is particularly marked when using just a few paths. This can be explained by the topology of the network. There are only a limited number of truly distinct paths between two endpoints available. As long as this number is not exhausted, the Sharability can be kept very small, but can also be large if a different metric for selection is chosen, hence the potential for deviation is large. Once all distinct paths are used up, additional paths have to include already used path segments. Hence the optimal value cannot be that small anymore, reducing the potential for deviation.

Using the Sharability as path selection criterion leads to an increase of the average path length between 15% and 20%. In the Shila-Measurement, this increase has no negative influence on the performance achieved.

[2]Remember, the MTU is identical for all paths in the network.

Paths	MTU	Path length	Sharability
Path Selection: MTU			
1	-	0	0
2	-	0	0.72 ± 0.21
4	-	0	0.34 ± 0.16
6	-	0	0.12 ± 0.08
8	-	0	0.06 ± 0.06
Path Selection: Path length			
1	0	-	0
2	0	-	0.72 ± 0.21
4	0	-	0.34 ± 0.16
6	0	-	0.06 ± 0.05
8	0	-	0.05 ± 0.05
Path Selection: Sharability			
1	0	0	-
2	0	0.21 ± 0.10	-
4	0	0.21 ± 0.07	-
6	0	0.16 ± 0.07	-
8	0	0.16 ± 0.07	-

Table 6.4: Percentage deviation to the optimal value of the other metrics for each path selection. Note that the Sharability for a single path is always zero. As an example of how to read the table: If the path selection is done using Sharability, the average length for two and four paths increases by 21% compared to the optimal length. The optimal length is determined by the path selection using path length as an optimization criterion.

Comparison with QUIC over SCION

Table 6.5 shows the values for goodput and throughput obtained from the Shila-Measurement and the Quic-Measurement.

Goodput

Concerning goodput, QUIC over SCION outperforms Shila. Compared to MPTCP via SCION with a single path, QUIC achieves an average goodput that is more than three times better. With the use of eight paths for MPTCP via SCION the performance is still more than 2.5 times better. The data exchange with MPTCP via SCION means a detour for the data in both endpoints. The individual segments travel via a virtual interface to the kernel, back to the userland application Shila and from there through SCION and its networking stack. Once at the destination, the same procedure is repeated in reverse order. These additional distances inevitably lead to an increase of the Round-Trip-Time which has a negative effect on the achieved goodput.

Throughput

About a third of the total throughput of MPTCP over SCION is overhead. For QUIC over SCION it is only around 10%, meaning that the latter approach performs better in this category as well.

Paths	Shila-Measurement		Quic-Measurement	
	Goodput	Throughput	Goodput	Throughput
1	10.75 ± 3.28	17.45 ± 5.91	33.31 ± 3.28	36.82 ± 3.62
8	12.74 ± 5.53	19.37 ± 8.36	-	-

Table 6.5: Comparison of goodput and throughput in MBits/sec between the Shila-Measurement and Quic-Measurement. For Shila, path selection was done by minimizing the path length.

6.4 Conclusion

Based on the results obtained, the stated questions can be answered as followed:

1. How does the performance behave in relation to the number of paths used for a connection?

   Increasing the number of paths leads to better performance. In the mea-

surements conducted an increase in average goodput up to 20% was observed.

2. How does the path selection influence performance?

   Using different path selection algorithms does not influence performance. For the measurements carried out, none of the selection criteria was able to bypass a potential bottleneck in the network infrastructure and thus achieve better performance than the other criteria.

3. How well does Shila compare to QUIC over SCION?

   QUIC over SCION outperforms MPTCP over SCION concerning goodput and overhead.

Although the questions can be answered based on the taken measurements, a final answer to the performance of Shila should not be derived from them. Rather, these initial results should be used as the basis for further questions and improvements to Shila. In Chapter 7, we discuss the questions that have emerged and possible improvements that can be addressed in future work.

Chapter 7

Future Work

In this chapter, we take up different possibilities and approaches to how the presented work around Shila can be improved and extended. The discussion refers to the results that were obtained in the course of this work and intends to provide the best possible basis for the further development of Shila.

Further Testing

After completing the implementation, we checked and ensured the basic functionality of Shila with tests both in local and SCIONLab infrastructure. However, during the execution of the Shila-Measurement, as presented in the previous chapter, individual experiments occasionally failed. Of course, such experiments were repeated and not included in the measurements. The exact reasons for the failure have not been investigated in the course of this work, but the failed experiments have all been documented. An important part of the revision and further development of Shila is to use this documentation to identify and fix possible weaknesses and bugs. Hereby, the infrastructure developed for the execution of the measurements can be optimally used to perform further extensive test runs.

Further Measurements

One goal of the measurements was to determine the influence of the path selection algorithm on the performance of Shila. Within the SCIONLab network, no influence could be detected. This is also due to the fact that the values of the metrics for different path options did not differ, e.g. all paths had the same value for MTU. For more comprehensive statements about the influence of the path selection further measurements are recommended. It is thereby important to consider whether the metrics used are the right ones for performance optimization or whether alternatives might be more suitable. For example, not only criteria relating to the static state of the network are conceivable. An interesting direction for further measurements could include investigations about the existence and influence of more dynamic criteria, offering more insight about the current health of the network than its static topology. Of course, the topology of the network also plays a decisive role in the investigation. If a topology offers a greater variety of different paths, it is easier to determine a possible influence of the path selection.

More flexible Routing-Information

In the current version of Shila, there is only one way to pass the Routing-Information, i.e. the mapping from TCP address to SCION address. The information has to be passed along with the startup of the shim layer. This assumes that the mapping is already known and brings along a restriction in flexibility.

One possible solution was investigated in the conceptual phase of the thesis: The record route option of the IP protocol allows to log the route of a datagram through the Internet. When enabled, each IP datagram has space in its header to store up to eight IP addresses that it encounters on its way from sender to receiver. A predefined data structure has to be specified during the creation of the socket to activate this non-compulsory socket option. This structure describes the 32 bytes of memory required for storing the addresses and can instead of the usual zero also be initialized with the SCION target address. Each packet sent from the socket created with this option contains the SCION destination address in its IP header. The SCION target address for a new Main-Flow can then be extracted by Shila from the first intercepted datagram. With a wrapper around the standard TCP API, the process of additional address specification can be made pleasant for the user. Of the 32 bytes of the record route option, 28 bytes can be used effectively. The first four bytes are overwritten when traversing the virtual interface on the way to Shila. A SCION address requires a maximum of 20 bytes. With this approach its therefore possible to pass further control information, up to eight bytes, from the application to Shila. At first glance, this approach has the disadvantage of increasing the overhead. But there is nothing to be said against reassembling the IP packet in Shila without the additional option. The stated approach is not yet part of the implementation since it is not strictly needed for the basic functionality. But once part of Shila, it increases the flexibility and possibilities of the shim layer.

Reducing Overhead

As stated in the previous chapter, the data exchange with MPTCP over SCION comes with comparatively high overhead. This is also due to the multiple nesting of the payload within a SCION packet: It contains a Shila-

Message[1], which itself holds a TCP segment in an IP datagram. The actual data sits inside the TCP packet. In this nesting, information which does not necessarily has to be transmitted is sent along. It is not mandatory to include the source and destination address in the IP header, these are defined by the TCP-Flow of the Backbone-Connection. The same applies to the protocol identification number or the checksum, which can be recalculated. An approach to reduce overhead would therefore be going without the IP header but just the TCP segment as payload and a custom header. Such a header holds only the information required to reassemble the IP packet at the receiving endpoint. An IP header without additional options has a size of 20 bytes. The identification value (2 bytes) and the flags together with the fragment offset (2 bytes) must be included in the custom header for every IP datagram processed by Shila. The remaining information necessary for the re-composition of the IP packet at the destination endpoint can either be calculated, e.g. the checksum, or is given through to the Backbone-Connection, e.g. source and destination address. With this approach, the contribution of the IP header to overhead can be reduced by 80%. Redundant information is also sent in the TCP segment, the source and destination ports are also given by the Backbone-Connection. But the savings potential for combining the TCP header is lower, as the majority of the fields contain dynamic information. If the proposal under discussion is implemented, the additional effort for parsing and recompiling the IP datagrams should not be disregarded.

No Namespaces

In the current implementation of Shila, the ingress interface is always assigned to the same IP address. Any TCP client starting a new data exchange via MPTCP over SCION specifies this address as the destination for the connection. Since the ingress and egress interfaces are per default located in a different namespace, this is fine. The client's traffic is routed to one of the dedicated egress interfaces and intercepted by Shila from where it is processed as intended. But if the interfaces are not separated, the traffic is routed only locally. Since the ingress interface matches the destination address (specified by the TCP client on the same host), the traffic is never about to be sent out via an interface and therefore cannot be intercepted by Shila. One possible approach to solve this problem is to adjust the local routing. We have decided against this intervention into the basic setup of the host. Another approach would be to extract the address (for the ingress interface) from the host SCION address. However, for hosts from different ASes, this is not necessarily different and therefore the problem is not solved. With the current implementation, Shila only works without namespaces if a unique

[1] A Shila-Message is the message sent between two Shila instances via the Backbone-Connection.

address for the ingress interface is configured on each host by hand. On a larger scale, this approach is not suitable. It is therefore required to invest further in a version of Shila that works without namespaces and does not require a manual assignment of IP addresses.

Side-by-side

To conclude this chapter, we present an approach how the traditional Internet and SCION can be used side-by-side using Shila and MPTCP. In contrast to the use case presented in this thesis, the side-by-side approach realizes the main-flow of an MPTCP connection via the conventional Internet. The SCION network is only used for possible sub flows. With such an approach the establishment of SCION could be further advanced. The new technology will become better known and will ideally contribute to improved user experience without interfering with a users (and applications) known working environment. Below we outline, supported by the illustration in Figure 7.1, how such a side-by-side solution would work.

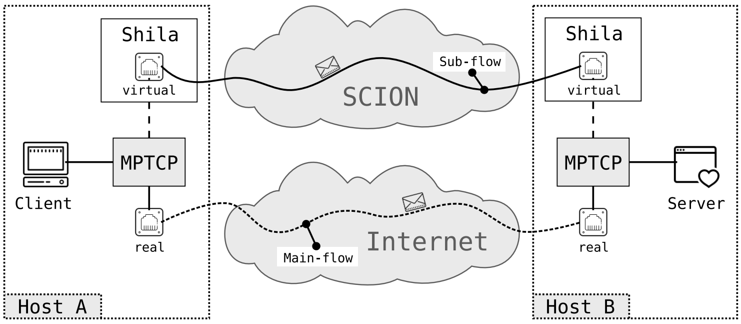

Figure 7.1: Illustration of the side-by-side approach. The main-flow of an MPTCP connection goes through the Internet, whereas the sub-flow (possibly multiple) goes through the SCION network. The user will not experience any change in his usual working environment, but can still benefit from any possible advantages of SCION.

We assume the following starting position: Two hosts, A and B, both part of the conventional Internet but also part of the SCION network. For access to the conventional Internet, the hosts have a corresponding real network interface. In addition, both hosts run Shila without using namespaces for their virtual network interfaces. On Host A, a client application now initiates a connection with the server application on Host B. MPTCP establishes a main-flow over the two real interfaces. To ensure that the initial connection is established over the real interface, the client specifies the appropriate interface when connecting.[2] As soon as the main-flow is established, MPTCP starts trying to connect sub-flows between all the different pairs of interfaces. The Shila instance on Host A intercepts the sub-flow connection tries[3] coming from its virtual interface. For the one directed towards the virtual interface of Host B, it can now create a backbone connection with the Shila instance listening there. The MPTCP connection now consists of a main-flow through the conventional Internet supplemented by a sub-flow through the SCION network. The application can only benefit from this; the mechanisms of MPTCP ensure that the availability of multiple paths is optimally used.

[2] For example by specifying the interface in the *connect(..)* call.

[3] MPTCP also tries to establish a sub-flow between the two virtual - real interface pairs. These attempts are ignored by the respective Shila instances after any useful information has been extracted.

Chapter 8

Conclusion

With the presented implementation of Shila, a shim layer between MPTCP and SCION, the combined use of the two promising technologies becomes possible. TCP applications can now be operated via SCION without any changes being necessary. If the endpoints furthermore support MPTCP, the applications also benefit from the advantages of using multiple paths orchestrated by MPTCP and inherently supported by SCION. With its prototype character, the implementation of Shila does not achieve the same performance as comparable approaches yet. However, with the development and improvement options we have identified for Shila, we were able to illustrate the potential of this approach and provide the groundwork for future research dedicated to close the performance gap. Even if it is only a tiny piece of the mosaic in SCION's ultimate goal: We are pleased with the contribution made in developing Shila for a more reliable, fairer and safer internet.

Bibliography

[1] Tom Anderson, Ken Birman, Robert Broberg, Matthew Caesar, Douglas Comer, Chase Cotton, Michael Freedman, Andreas Haeberlen, Zachary Ives, Arvind Krishnamurthy, William Lehr, Boon Loo, David Mazières, Antonio Nicolosi, Jonathan Smith, Ion Stoica, Robbert Van Renesse, Michael Walfish, Hakim Weatherspoon, and Christopher Yoo. The nebula future internet architecture. pages 16–26, 05 2013.

[2] Sébastien Barré, Christoph Paasch, and Olivier Bonaventure. Multipath tcp: From theory to practice. In Jordi Domingo-Pascual, Pietro Manzoni, Sergio Palazzo, Ana Pont, and Caterina Scoglio, editors, *NETWORKING 2011*, pages 444–457, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[3] M. Handly D. Wischik Univ. College London C. Raiciu, Univ. Politehnica of Bucharest. Coupled congestion control for multipath transport protocols. RFC 6356, RFC Editor, 10 2011.

[4] Université catholique de Louvain. Uclouvain strengthens connectivity of 1 in 8 of world's smartphones. `https://www.eurekalert.org/pub_releases/2019-11/ucdl-usc111819.php`. Accessed: 2020-08-13.

[5] Yung-Chih Chen, Yeon-sup Lim, Richard Gibbens, Erich Nahum, Ramin Khalili, and Don Towsley. A measurement-based study of multipath tcp performance over wireless networks. *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, 10 2013.

[6] Quentin Coninck and Olivier Bonaventure. Multipath quic: Design and evaluation. pages 160–166, 11 2017.

[7] Quentin De Coninck. *Flexible Multipath Transport Protocols*. PhD thesis, UCLouvain, March 2020. Presentation available at https://www.youtube.com/watch?v=b2FWfT9b3Ws&feature=youtu.be&t=132.

[8] Jonathan Corbet. Accelerating networking with afxdp. https://lwn.net/Articles/750845/. Accessed: 2020-08-19.

[9] DigitalOcean. Digitalocean. https://www.digitalocean.com/. Accessed: 2020-07-22.

[10] Michael A. Flückiger. Shila - implementing and evaluating mptcp on the scion future internet architecture. https://github.com/fluckmic/shila. Accessed: 2020-07-22.

[11] A. Ford, C. Raiciu, et al. Tcp extensions for multipath operation with multiple addresses. RFC 793, RFC Editor, 1 2013.

[12] Linux Foundation. Data plane development kit. https://dpdk.org/. Accessed: 2020-08-19.

[13] Wireshark Foundation. tshark - dump and analyze network traffic. https://www.wireshark.org/docs/man-pages/tshark.html. Accessed: 2020-08-11.

[14] Sana Habib, Junaid Qadir, Anwaar Ali, Durdana Habib, Ming Li, and Arjuna Sathiaseelan. The past, present, and future of transport-layer multipath. *Journal of Network and Computer Applications*, 75, 01 2016.

[15] Scott Hogg. What is sctp and why are we not using it? https://www.networkworld.com/article/2222277/what-about-stream-control-transmission-protocol--sctp--.html. Accessed: 2020-08-13.

[16] C. Huang, M. Lin, and L. Chang. The design of mobile concurrent multipath transfer in multihomed wireless mobile networks. *The Computer Journal*, 53(10):1704–1718, 2010.

[17] Toke Høiland-Jørgensen, Jesper Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. The express data path: fast programmable packet processing in the operating system kernel. pages 54–66, 12 2018.

[18] L. Xu UNL S. Ha Colorado A. Zimmermann L. Eggert R. Scheffenegger NetApp I. Rhee, NCSU. Cubic for fast long-distance networks. RFC 8312, RFC Editor, 2 2018.

[19] OpenVPN Inc. Qpenvpn. https://openvpn.net/. Accessed: 2020-08-13.

[20] louvain-la-neuve belgium ip network lab, ucl. Multipath tcp - linux kernel implementation. https://multipath-tcp.org/. Accessed: 2020-07-22.

[21] J. R. Iyengar, P. D. Amer, and R. Stewart. Concurrent multipath transfer using sctp multihoming over independent end-to-end paths. *IEEE/ACM Transactions on Networking*, 14(5):951–964, 2006.

[22] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-29, Internet Engineering Task Force, June 2020. Work in Progress.

[23] The kernel development community. Afxdp. https://www.kernel.org/doc/html/latest/networking/af_xdp.html. Accessed: 2020-08-19.

[24] Ramin Khalili, Nicolas Gast, Miroslav Popovic, and Jean-Yves Le Boudec. Opportunistic Linked-Increases Congestion Control Algorithm for MPTCP. Internet-Draft draft-khalili-mptcp-congestion-control-05, Internet Engineering Task Force, July 2014. Work in Progress.

[25] Maxim Krasnyansky. Universal tun/tab driver. http://vtun.sourceforge.net/tun/. Accessed: 2020-08-01.

[26] Maxim Krasnyansky. Vtun. http://vtun.info/. Accessed: 2020-08-13.

[27] John Kristoff. The transmission control protocol. https://condor.depaul.edu/~jkristof/technotes/tcp.html. Accessed: 2020-08-12.

[28] Scott Lowe. Introducing linux network namespaces. https://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/. Accessed: 2020-08-08.

[29] ICSI-E. Blanton Purdue University M. Allman, V. Paxson. Tcp congestion control. RFC 5681, RFC Editor, 9 2009.

[30] Björn Töpel Magnus Karlsson. The path to dpdk speeds for afxdp. http://vger.kernel.org/lpc_net2018_talks/lpc18_paper_af_xdp_perf-v2.pdf. Accessed: 2020-08-19.

[31] ETH Zurich Network Security Group. appnet - a simplified and functionally extended wrapper interface to the scionproto/scion package snet. https://github.com/netsec-ethz/scion-apps/tree/master/pkg/appnet. Accessed: 2020-08-08.

[32] ETH Zurich Network Security Group. scion - scalabiltiy, control, and isolation on next-generation networks. https://www.scion-architecture.net/. Accessed: 2020-07-20.

[33] ETH Zurich Network Security Group. Scionlab - a global research network to test the scion next-generation internet architecture. https://www.scionlab.org/. Accessed: 2020-07-26.

[34] ETH Zurich Network Security Group. videos about the scion internet architecture. https://www.scion-architecture.net/pages/videos/. Accessed: 2020-07-20.

[35] Christoph Paasch, Gregory Detal, Fabien Duchene, Costin Raiciu, and Olivier Bonaventure. Exploring mobile/wifi handover with multipath tcp. *CellNet'12 - Proceedings of the ACM Workshop on Cellular Networks: Operations, Challenges, and Future Design*, 08 2012.

[36] Adrian Perrig, Pawel Szalachowski, Raphael M. Reischuk, and Laurent Chuat. *SCION: A Secure Internet Architecture*. Springer Publishing Company, Incorporated, 1st edition, 2017.

[37] J. Postel. Transmission control protocol (tcp). RFC 793, RFC Editor, 9 1981.

[38] The Chromium Projects. Quic, a multiplexed stream transport over udp. https://www.chromium.org/quic. Accessed: 2020-08-12.

[39] Ed. R. Stewart. Stream control transmission protocol. RFC 4960, RFC Editor, 9 2007.

[40] C. Raiciu, S. Barré, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath tcp. In *SIGCOMM 2011, Toronto, Canada*, August 2011. See http://www.multipath-tcp.org for related work on MPTCP and the Linux kernel implementation used in this paper.

[41] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? designing and implementing a deployable multipath tcp. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, page 29, USA, 2012. USENIX Association.

[42] Daniel Stenberg. Tcp head of line blocking. https://http3-explained.haxx.se/en/why-quic/why-tcphol. Accessed: 2020-08-13.

[43] ubuntu manuals. ip-netns - process network namespace management. http://manpages.ubuntu.com/manpages/trusty/man8/ip-netns.8.html. Accessed: 2020-08-08.

[44] Anwar Walid, Qiuyu Peng, Jaehyun Hwang, and Steven H. Low. Balanced Linked Adaptation Congestion Control Algorithm for MPTCP. Internet-Draft draft-walid-mptcp-congestion-control-04, Internet Engineering Task Force, January 2016. Work in Progress.

[45] Wikipedia. Keepalive. https://en.wikipedia.org/wiki/Keepalive. Accessed: 2020-08-09.

[46] Wikipedia. Stream control transmission protocol (sctp). https://en.wikipedia.org/wiki/Stream_Control_Transmission_Protocol. Accessed: 2020-08-13.

[47] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath tcp. pages 8–8, 03 2011.

[48] XDP-project. The express data path (xdp) inside the linux kernel. https://github.com/fluckmic/shila. Accessed: 2020-08-19.

[49] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David Andersen. Scion: Scalability, control, and isolation on next-generation networks. pages 212 – 227, 06 2011.

[50] S.Q. Zheng, Bing Yang, Mei Yang, and Jianping Wang. Finding minimum-cost paths with minimum sharability. pages 1532 – 1540, 06 2007.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

IMPLEMENTING AND EVALUATING MPTCP ON THE SCION FUTURE INTERNET ARCHITECTURE

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Flückiger	Michael Alexander

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Aadwil, 25.08.2020

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.