Otto-von-Guericke-University Magdeburg
Faculty of Computer Science
Networks and Distributed Systems Lab

Master Thesis

Improving SCION Bittorrent with efficient Multipath Usage

Version 1.0, November 2020

Date: November 11, 2020

By: Marten Gartner

Supervisors: Prof. Dr. David Hausheer
Prof. Dr. Mesut Güneş

Advisors: Martin Koppehel
Thorben Krüger

Abstract

This thesis extends an existing Bittorrent over SCION implementation to improve download performance by aggregating multiple paths to each available peer. To achieve this goal, peers are identified by a combination of their address and the path used to transfer packets to them, called *path-level* peers. This approach allows Bittorrents thoroughly researched file sharing algorithms to run on path level, instead of implementing a dedicated multipath connection to each peer.

The multipath design consists of two core steps: The gathering of path-level peers out of peer addresses and the determination of a set of paths to use for each peer. For each peer, up to a static number of path-level peers are connected to avoid excessive growth. The decision from which path-level peers the actual download proceeds is designed by measuring the handshake time as well as measured path bandwidths.

To evaluate the presented multipath implementation, a comparison of various SCION candidates is performed. To locate possible bottlenecks in the SCION stack, the multipath implementation is evaluated with running a high performance SCION router as well as using jumbo frames to improve download bandwidth. The evaluation setup targets the download over multiple, high performance paths to a single peer, to be later scaled to larger topologies.

Compared to the existing SCION singlepath approach, a performance improvement of around 300% is observed by aggregating paths and using jumbo frames. This improvement is mainly caused by using jumbo frames, since an additional bottleneck in the SCION stack limits the performance increase of aggregating multiple paths. In case of path congestion, the multipath implementation is able to shift bandwidth to the free path reaching constant bandwidth. Finally, the maximum performance of Bittorrent over SCION is evaluated using a varying number of torrents and different torrent sizes.

Contents

List of Abbreviations

SCION Scalability, Control, and Isolation on Next Generation Networks

TCP Transmission Control Protocol

UDP User Datagram Protocol

µTP Micro Transport Protocol

QUIC Quick UDP Internet Connections

MTU Maximum Transmission Unit

DDoS Distributed Denial of Service

P2P Peer-to-Peer

HTTP Hypertext Transfer Protocol

URL Uniform Resource Locator

IOT Internet Of Things

OSI Open Systems Interconnection

IP Internet Protocol

DNS Domain Name System

SDN Software-defined Networking

AS Autonomous System

ISD Isolation Domain

TRC Trust Root Chain

PCB Path-segment Contruction Beacon

MAC Message Authentication Code

DPDK Dataplane Development Kit

VPP Vector Packet Processing

CPU Central Processing Unit

HSR High Speed Router (SCION)

JF Jumbo Frames

TLS Transport Layer Security

AP Attachment Point

NIRA New Internet Routing Architecture

SIBRA Scalable Internet Bandwidth Reservation Architecture

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Although the Internet was designed decades ago, many of the original shortcomings persist to this day. From the perspective of a client, no control over the path that is used to access a resource over the Internet is possible. Furthermore, there is no guarantee that every packet sent from a given source to a given destination will always use one particular path. Finally, there is no global approach for using multiple different paths to a specific destination.

Researchers proposed different approaches to overcome these drawbacks within the limits of the current Internet architecture [1, 2]. However, some recently started working on redesigning the complete Internet architecture from scratch to avoid the old mistakes. Wile the approaches [3–5] have their own merits, this work focuses on SCION introduced by Perrig et al [6]. SCION is an acronym for *Scalability Control and Isolation On Next Generation Networks*. This architecture provides full and transparent path control for hosts as well as built-in defense against hijacking or DDoS attacks and trust issues. Along with path control, it allows using multiple paths to the same remote destination, which offers new possibilities on the application level. Although the number of applications with SCION support is steadily increasing, their use in research is still in its infancy. In this work, the widely used Bittorrent application protocol will be made path aware through SCION and gain multipath support. As an application protocol, Bittorrent has attracted much attention from the research community, usually with setups containing a large number of peers and expansive network topologies [7, 8], often resulting in low performance. This work aims to target a different setup, where fewer peers and high performance links are combined into simpler topologies, in order to measure how a multipath-enabled Bittorrent application performs on high bandwidth SCION links.

1.2 Goal

This thesis covers the design, implementation and evaluation of a multipath extension to the current Bittorrent implementation for SCION [9] and addresses the following research question: To what extent can the SCION path-aware architecture improve the existing

Bittorrent implementation for use over high performance links? To answer this question, the implementation is evaluated focusing on two core aspects: 1) Bandwidth aggregation over multipath and 2) optimized path selection. The goal of this thesis is to show how the use of built-in SCION functions like path-awareness can improve the existing Bittorrent implementation with respect to download performance.

1.3 Structure

The rest of this work is organized as follows. The next chapter provides background to understand the presented concepts and ideas, focusing on SCION and Bittorrent. In chapter 3, the current state of the existing Bittorrent over SCION implementation is discussed. Afterwards, the multipath extension is presented, separated into the implementation of 1) multipath bandwidth aggregation and 2) optimized path selection. In chapter 4 the benchmark setup and parameters that are used in the evaluation are presented. Afterwards, chapter 5 covers the evaluation of the finished multipath Bittorrent over SCION implementation. Finally, related work is presented in chapter 6 followed by a conclusion and outlooks for future work in chapter 7.

2 Background

This chapter addresses the required background to understand this work, mainly focused on Bittorrent and SCION fundamentals.

2.1 Bittorrent

Bittorrent is a P2P protocol with many implementations in different programming languages which aims to allow fast and efficient file distribution by leveraging the upload capacity of downloading peers. While a peer downloads a file, it makes any parts it has already retrieved available to other peers to contribute its upload capacity to the distribution of the file.

2.1.1 Concepts

Through the P2P approach, Bittorrent aims to overcome the limits of classic client-server setups, where a server can be a single source of failure The Bittorrent protocol specifies file transfer as a distributed mechanism between peers without the need for central coordination Nevertheless, some initial coordination between peers is still necessary to exchange contact information. Bittorrent specifies one approach for this information exchange via a so called *tracker*. A tracker is typically accessed via HTTP. To retain the P2P benefits of Bittorrent, the tracker only acts as resource for meta information, the files themselves are exchanged between the peers directly. In Bittorrent, files are not transferred as bytestream containing the complete file. Instead, the basic idea behind Bittorrent is to divide large files into equal-sized *pieces* (typically with a size between 32KB and 256KB).These pieces are exchanged between peers in parallel. Bittorrent peers can either act as *seeders*, which have a complete copy of a specific file which they share with to others, or as *leechers*, that that do not yet have a complete copy but share any pieces of the file they have already obtained. For each file provided by the system there must be at least one seeder, that initiates the distribution of the file. Figure 2.1 shows the process of downloading a file in Bittorrent.

At the beginning of the download process, the *tracker phase*, the requesting peer asks the tracker where to find the pieces for the desired file. In response, the tracker returns a list of

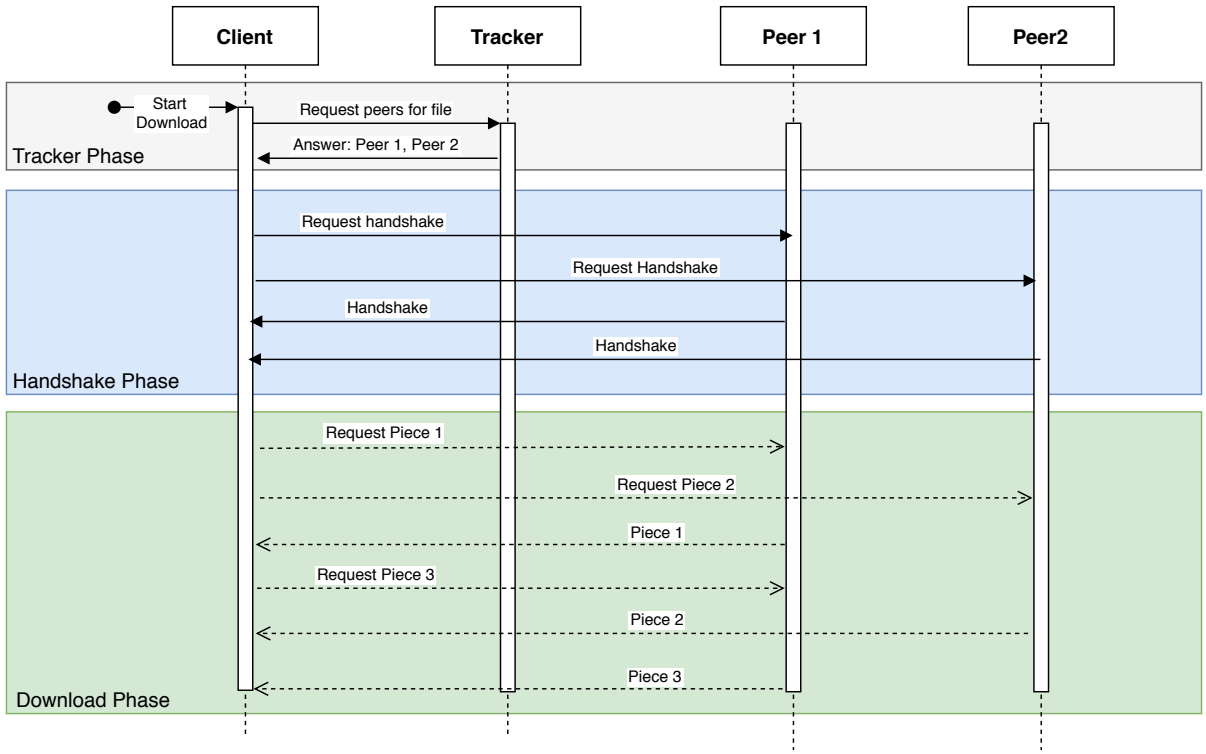

Figure 2.1: Process of downloading a file in Bittorrent. As seeders, Peer1 and Peer2 are currently not choking.

random peers which provide pieces of the file. Next, in the *handshake phase* the requesting peer attemps to establish a connection to each returned peer. After completing the handshakes, the peer starts requesting pieces over the opened connections. The download process is complete if all pieces are completely downloaded on the requesting peer.

To represent the required information to download a file over the Bittorrent protocol, *meta info files* were introduced. Individual pieces of the file are referenced together with their SHA-1 hash in a meta info file ending with the .torrent suffix, which contains all information required to download the file using a Bittorrent client. Figure 2.2 shows the content and the structure of a .torrent file.

The content of a meta info file is a bencoded [10] dictionary containing different fields. It starts with an announce field, which contains the announce URL of the tracker, followed by an info dictionary. This dictionary differs if a single file or multiple files are referenced. In both cases, the info dictionary contains the piece length field indicating the number of bytes in each piece and the pieces array, which contains SHA-1 hashes of each piece. In single file mode, also the name and length fields are included, which contain the suggested file name and the length of the file in bytes, respectively. In multiple file mode, the name field contains the directory name where to store all files and a files field, which encodes path and length tuples for each contained file.

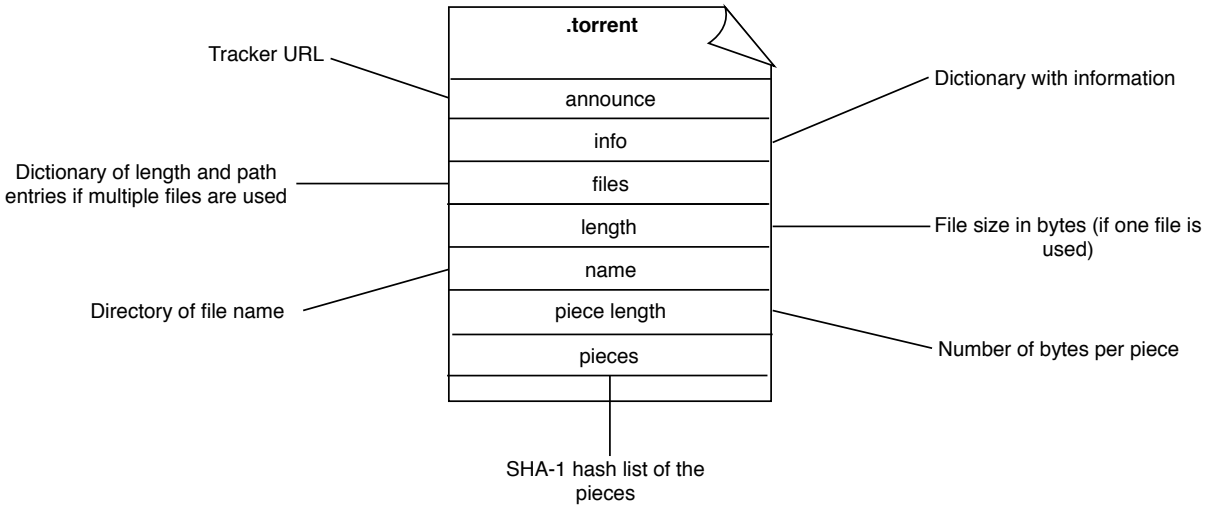

Figure 2.2: Structure of a .torrent meta info file.

2.1.2 Piece Selection

Random First

At the beginning of the download, the peer has no pieces of its own to contribute to the distribution of the file. In order to address this as quickly as possible (and avoid getting choked), the peer starts requesting random pieces in parallel. When the first piece has been fully retrieved, the peer transitions to the rarest first strategy.

Rarest First

The rarest first selection strategy aims increase distribution of those pieces, that are not provided by many peers by prioritizing their retrieval and subsequent dissemination to other peers. This strategy also aims to reduce the risk of having a particular piece no longer available, in case of e.g. all seeders leaving the network prematurely.

Endgame Mode

Some peers may have significantly lower bandwidth than others. This is not a problem during highly parallel retrieval in the middle of a download, but can significantly reduce the download bandwidth when only a few pieces remain to be retrieved. Consequently, Bittorrent also specifies an endgame mode, in which all remaining pieces are requested from all peers in parallel. Once the specific piece is completed, the peer sends cancel requests to the remaining conencted peers. Since the endgame mode is very short in practice, this approach leads to faster download completion without much bandwidth overhead [11].

2.1.3 Pipelining

Requesting one piece after another would not be very efficient. Consequently, Bittorrent introduces pipelining to always have several requests pending at once. At introduction time of Bittorrent, around 5 requests were pipelined for each connection. The Bittorrent implementation used in this work pipelines 64 pieces per connection by default. After each retrieved piece, a new piece request is pipelined.

2.1.4 Choking Algorithms

Generally, a peer is mainly interested in maximizing its own download rate, which is however directly dependent on the upload bandwidth contribution of the other peers. As a P2P system, Bittorrent does not have centralized resource management, there is no central mechanism to control the download and upload rate of peers. To help creating a balance between the upload and download rate of peers, Bittorrent instead encourages a variant of *tit-for-tat*: A peer can decide to cooperate with or to *choke* other peers. Choking is equivalent to a temporary refusal to a certain peer. Each Bittorrent peer always unchokes a fixed number of peers (per default 4). The choice of which peers to unchoke poses an interesting problem. In order to maximize the download rate, a peer wants to unchoke those peers providing the best upload rate to benefit from requesting pieces from them.

To decide which peers to unchoke, two different cases are possible. First, cyclic bandwidth measurements are performed to already connected peers. Based on the results, a decision is made to unchoke a peer or to keep it choked. For connecting to new peers, Bittorrent specifies an *optimistic unchoking* algorithm: Every 30 seconds, one random peer returned from the tracker will be unchoked, regardless of its bandwidth.

The choking and unchoking algorithms help to avoid free-riding and to create fairness within the network. The data transfer of pieces start whenever a leecher is interested, meaning it wants to obtain pieces of a particular file, and the peer is not choking. The interest state itself must be always kept up to date on each peer. Each connection starts choked and uninterested. After sending the interested flag to a peer, the respective connection may be unchoked and the download can begin.

2.2 SCION

SCION is a new Internet architecture designed to provide built-in functionalities tailored to the requirements of modern networks. The term SCION refers to the SCION protocol [6] as well as the SCION reference implementation [12] used in this work. SCION was conceived

to move beyond the stage of merely fixing problems of the current internet architecture and to enter into a stage where such problems can not occur by design. For example, SCION comes with built-in prevention against DDoS attacks, which caused a lot of economic damage over the last years [13], network and path transparency by design and build-in heterogeneous trust. Consequently, SCION offers possibilities to achieve highly available enterprise connectivity using redundant paths or to act as secure network architecture for IOT devices.

Figure 2.3 shows the scope of SCION in the OSI and TCP/IP network stack. SCION covers OSI layers 3-7 or for TCP/IP model layers 2-4, respectively. In the network/internet layer, SCION replaces inter-domain routing elements of IP, while still supporting IP address intra-domain. In the transport layer, SCION supports different transport protocols, with QUIC [14] as currently most popular choice for SCION applications. In the application layer, SCION provides built-in session cryptography and a dedicated DNS service called RAINS [15].

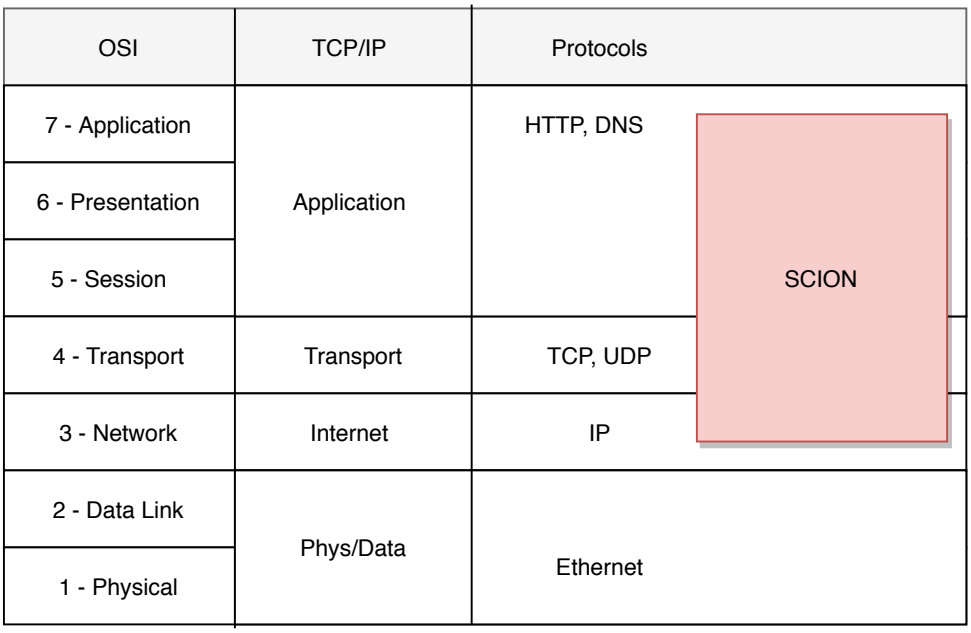

Figure 2.3: Location of SCION within the OSI and TCP/IP network stack

SCION comes with significantly different concepts compared to the current Internet architecture. It distinguishes between the control and the data plane. The control plane is responsible for routing and information exchange between network participants. The data plane is responsible for packet forwarding and path resolution. This division of responsibilities follows principles that are known from SDN architectures. In contrast to many modern SDN approaches [16], packet forwarding in SCION is still done in software, without dedicated hardware support. Furthermore, with the SCION packet header, state keeping within routers can be significantly reduced, while other SDN approaches need to keep e.g. flow tables on the routers.

2.2.1 Architecture

Networks running on SCION are separated into different isolation domains, called *ISDs*, which can be correspond to geographical regions, company networks or research networks. Each ISD contains one or more *ASes*, which contain of at least one SCION endhost. Within an ISD, typically more than one AS is operating as a *Core AS*. The Core ASes of one ISD are collectively referred to as the *ISD Core*. Together, the Core ASes build the trust root configuration *TRC*, which provides basic cryptographic information like secrets and keys, to which all other ASes must agree. Each AS has a private key and a certificate to authenticate itseld in different SCION processes on the control plane as well as the data plane. TRCs are negotiated between all core ASes of an ISD and build the base for all AS certificates.

Connections in the SCION architecture can be either native SCION links or overlay SCION links, which are tunneled across normal IP links. Links between Core ASes are referred to as *core links*. Usually, ASes are connected in a way to link them to Core ASes, either directly or via intermediate ASes. Direct links between two normal ASes that are not used to connect ASes to Core ASes are referred to as *peering links*. To interconnect ASes among each other, the *SCION Border router* comes into play.

Figure 2.4, shows an example SCION network structure. This example contains two ISDs *ISD 1* and *ISD 2*. Each of them contains at least one Core AS, *Core AS 1* and *Core AS 2* for *ISD 1* and *Core AS 3* for *ISD 2*. All regular ASes are directly or indirectly connected to their core ASes. Furthermore, *AS 6* and *AS 7* are connected with a peering link crossing the borders of both ISDs. As can be seen from Figure 2.4, a regular AS can have connections to more than one Core AS.

2.2.2 Control Plane

By distinguishing between data and control plane, SCION enables a clear separation of responsibilities for the different components. The control plane is responsible for routing decisions. Instead of collecting network information in large routing tables, SCION uses a so called *beaconing* mechanism. Beaconing is performed by SCION beacon servers operated by the core ASes, which periodically send path construction beans (*PCB*). Each PCB contains a creation timestamp. The SCION border router of each receiving AS adds its signature along with interface information to the PCB before forwarding it. PCBs are also exchanged between core ASes, a process called *core beaconing*. In the beaconing process, the TRC and the AS certificates are used to ensure AS authentication.

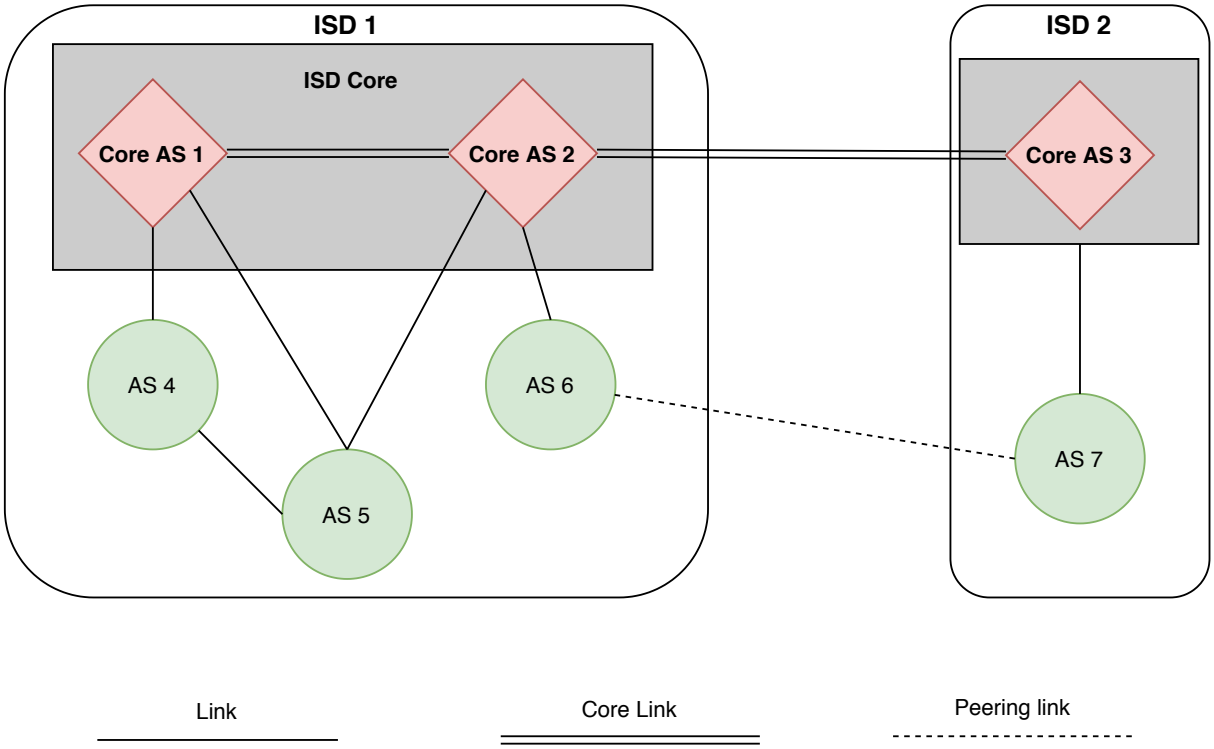

Figure 2.4: Example SCION network architecture consisting of two ISDs

PCBs contain one or more *path segments*. A path segment can either be a *up path segment* or a *down path segment*. Figure 2.5 shows the beaconing process. PCBs are created at the core ASes and are sent downwards to regular ASes. In case of *Core AS 1*, the created PCB is sent downwards over *AS 4* to *AS 5*. From the perspective of *Core AS 1*, this path segment is a down path segment, whereas for *AS 5*, it is an up path segment. SCION, up path and down path segments are invertible by reversing their order.

Having ASes interconnected among themselves mandates some form of loop handling in the beaconing process. SCION has several approaches to avoid beaconing loops. Core ASes avoid beacon loops ,both on the ISD and AS level. On the ISD level, any PCBs that already entered the ISD are discarded. On the AS level, any PCBs that already contain an entry representing that same AS is discarded. Regular ASes avoid beaconing loops by not forwarding PCBs to any ASes that already are part of the PCB.

Another important responsibility of the control plane is path lookup, where an AS queries available paths to a specific SCION address. A SCION address to a target host is represented as a triplet containing the ISD, the AS and the local address of the target host. Path lookup consists of two main steps, the name resolution and the lookup itself. SCION provides its own name resolution service called *RAINS*, which returns the SCION address of the requested name. With this address, a path lookup operation can be performed to retrieve path segments. The path lookup returns a triplet consisting of:

- *Up path segment*: Path segment from local AS to core AS

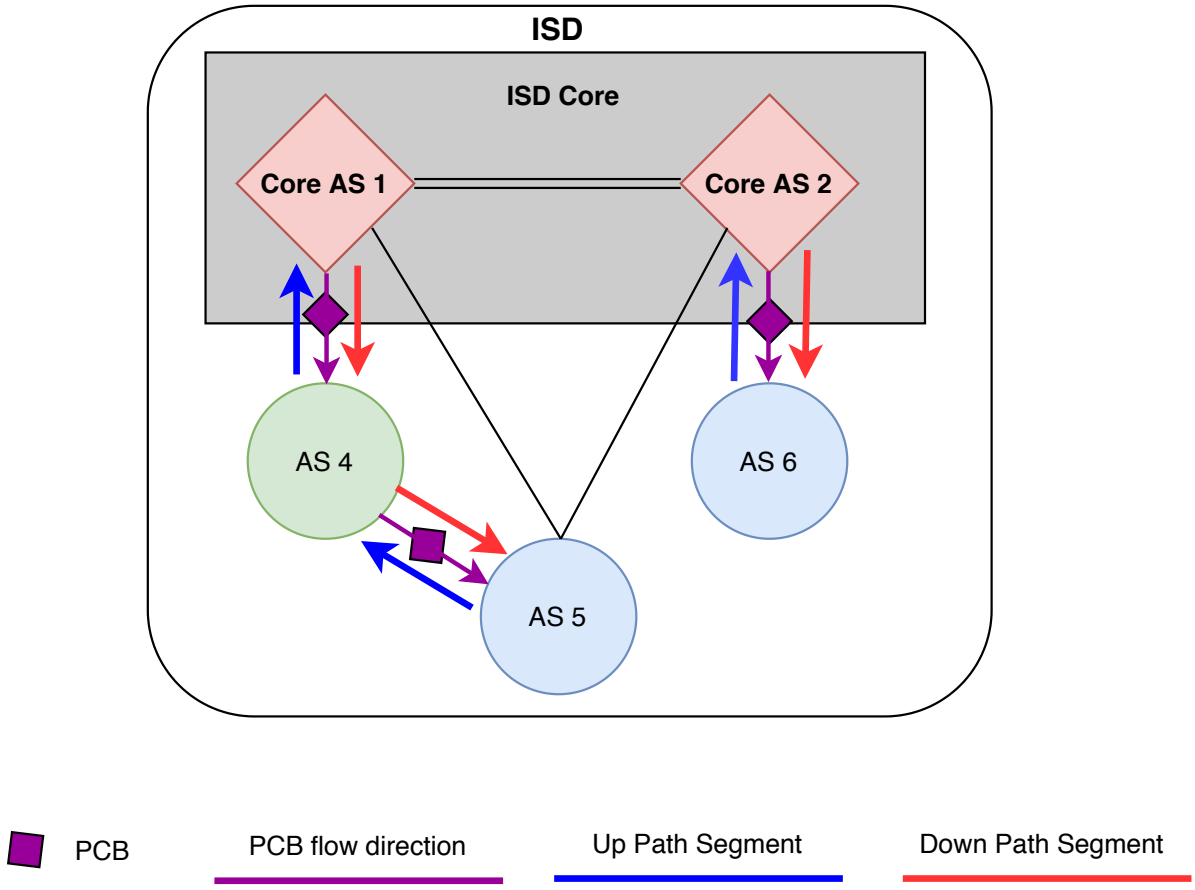

Figure 2.5: Up path and down path segments in PCBs

- *Core path segment*: Path segment from core AS to remote ISD core AS
- *Down path segment*: Path segment from remote core AS to remote AS

Any element of the path triplet may be omitted, depending on the network topology, e.g. a core path segment may not be required if a peering link exists on the path to the remote AS.

Figure 2.6 shows the path lookup process to a SCION AS in a remote ISD. In this case, AS 5 in ISD 1 performs a path lookup for AS 7 in ISD 2. At first, the local path server of the source AS is queried. This path server performs a lookup in its path cache (1). In case a valid path is found, the lookup process is now finished. Otherwise, the request is forwarded to the core path server of the next core AS 2 (2). Again a lookup in the path cache is done (3) and in case of a valid match, the path is returned to the source AS. Otherwise, the request is forwarded to the path server of core AS 3 (4). This core path server performs a local cache lookup (5) and returns the down-path segment to AS 7 back to core AS 2. Core AS 2 returns those down-path segment along with the up-path and core-path segment back to the path server of AS 5 (6). AS 5 now has the segments to construct a path to AS 7.

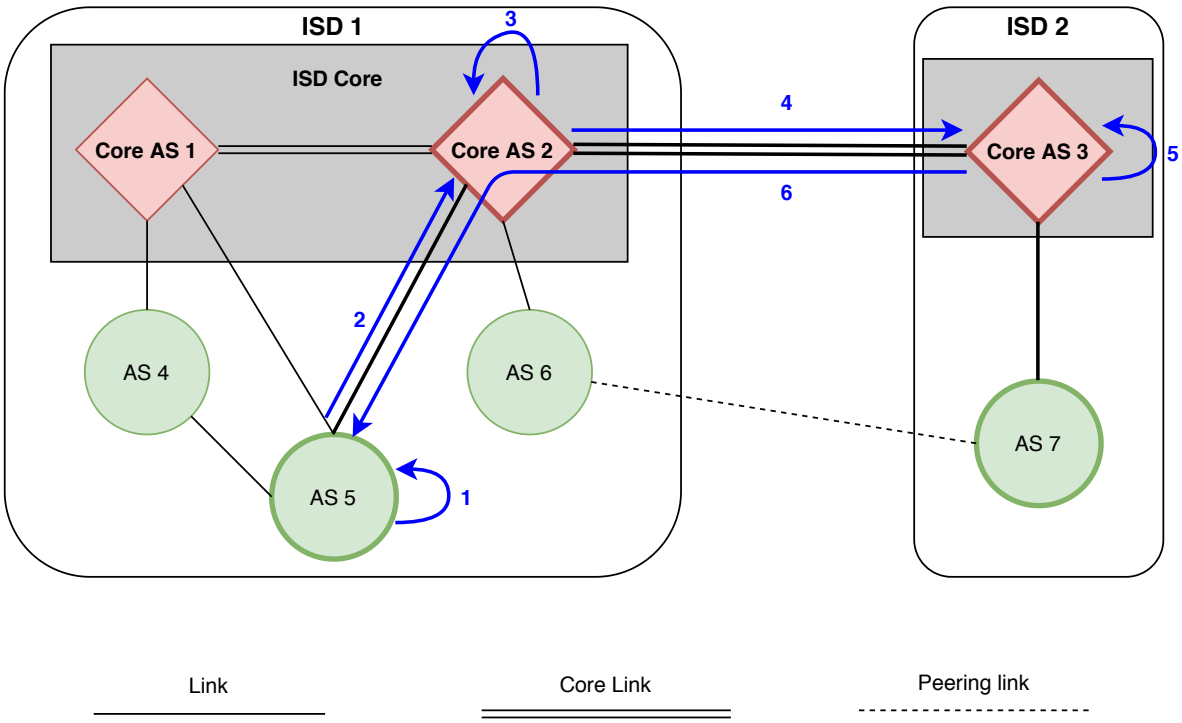

Figure 2.6: Path lookup process to a SCION AS in a remote ISD

2.2.3 Data plane

The main responsibility of the data plane is packet forwarding. To send packets to a remote destination, a path to this destination must be determined and encoded in the packets. Compared to the current internet architecture, this significantly reduces the state that routers have to keep, because the information where to forward the packet is stored in the packet itself. A path is defined as a combination of up path, down path and core path segments and is encoded in a special efficient format to reduce bandwidth overhead.

Figure 2.7a visualizes the path construction in a sample SCION network consisting of 6 ASes and 2 core ASes. In this example, AS *A* wants to send packets to AS *F*. Consequently, *A* starts a path lookup and retrieves the path segments depicted in Figure 2.7b , where the segment starting with info field *INF1* represents the up path segment, the one with *INF2* the core path segment and *INF3* the down path segment. Each path segment contains an ordered list of *hop fields*. Each hop field represents a description of physical or virtual links between two interfaces, the *ingress* and *egress* interface. Furthermore, the hop field contains the AS signature, the message authentication code *MAC*. Additional info fields are contained in each hop field. For up and down paths, the path segments start with the core AS and followed by the other ASes. Peering links are also encoded as hop field and returned from the path server, as can be seen in case of the link from AS *B* to *E*, With these path segments, one possible constructed path from A to F and reversed from F to A is shown in Figure 2.7c. (Note that since there is also a peering link, this is not

the shortest path in terms of the number of hops.) The AS A now orders the info fields containing the hop fields between its location in the network and the location of AS F, thereby creating a valid path to F. The Figure finally also shows the path in a reversed view from F to A containing the same hop fields.

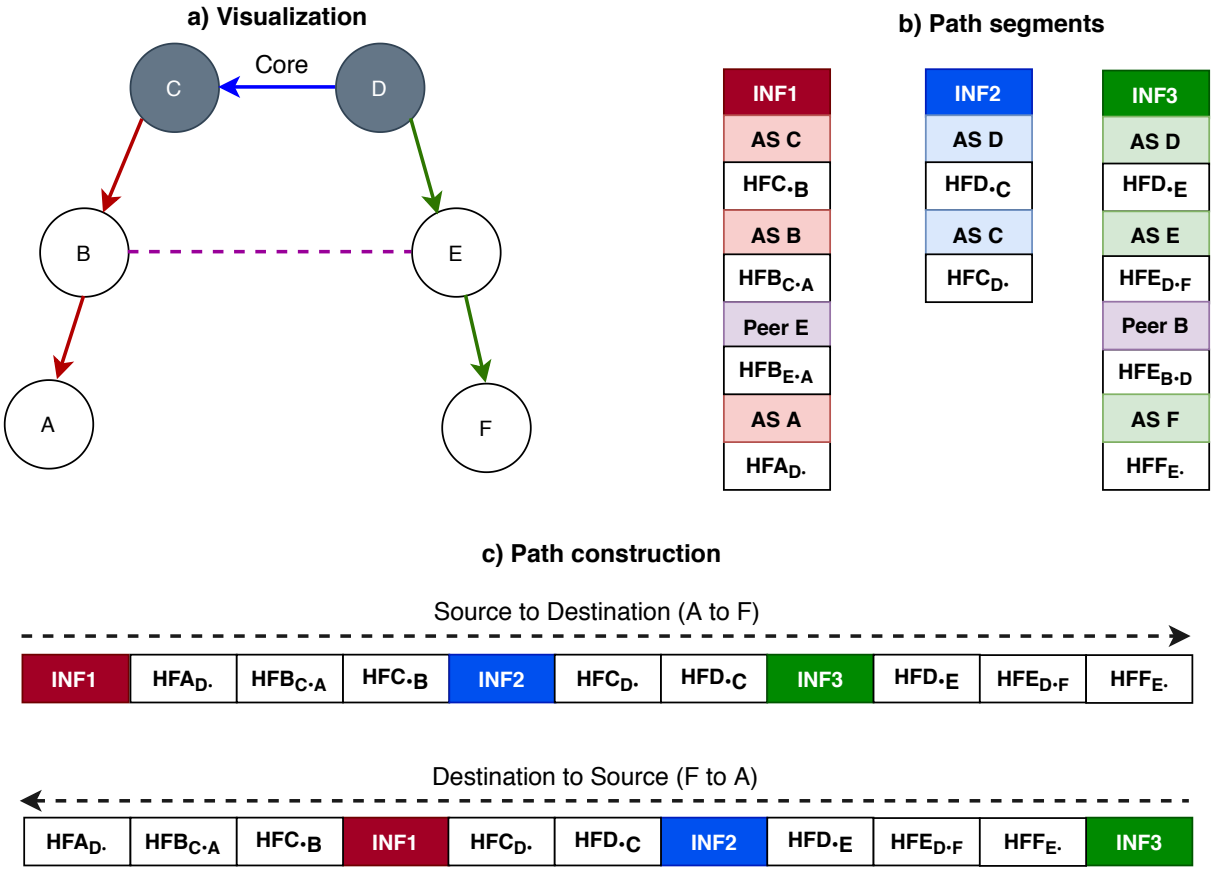

Figure 2.7: SCION path encoding with a network visualization in a), the respective path segments response from AS A querying paths to AS F in b) and constructed paths from A to F and vice versa in c).

Now containing a valid path in each SCION packet, the packets have to be forwarded. This is usually done by the SCION border router. The packet forwarding process differs significantly from the one in use in the current internet architecture, since the complete path is available in each packet. Consequently, the border router does not need to keep state about which SCION address should be forwarded over which interface, because the ingress and egress interfaces are already specified in the hop fields. To ensure the packet traverses the correct ASes that are referenced in the hop fields, each AS computes its MAC with its secret key and compares it against the MAC stored in the hop field. This verification is implemented through cryptographic algorithms, e.g. CMAC and AES and can be performed with a single encryption operation. Consequently, the computing effort is comparatively low. Using a common hardware based cryptographic engine this can be done in about 50 cycles on e.g. an Intel® Core™ i7 Processor Extreme Edition i7-980X [17, 18].

2.2.4 SCION Host Architecture

The SCION network architecture and the control and data plane responsibilities are implemented through a set of services running on a single SCION end host as well as on the AS level.

Figure 2.8 shows the components and services along with the necessary protocols to operate a SCION end host inside a SCION AS. Each SCION end host runs at least the *SCION daemon*, that is responsible for controlling the host, and the *SCION dispatcher*, which is responsible for parsing and preparing all outgoing packets. The AS operates additional services such the local path server and the SCION border router.

The presented host in this example runs an app that communicates over SCION. As mentioned before, SCION supports multiple transport protocols, like TCP, UDP or QUIC, which can be utilized by this app. All sent and received packets of the app that run over SCION are passed to the SCION dispatcher, which converts them into SCION packets, using a suitable transport protocol. The communication to the dispatcher is implemented in two different ways, depending on the transport protocol used. TCP packets are passed to the TCP stack of the dispatcher. UDP packets are transferred via a SCION socket library to the dispatcher and vice versa. Outgoing SCION packets are then sent to the SCION border router of the AS, which forwards them to the next AS. Incoming SCION packets are handled by the border router and passed to the dispatcher. Control plane features are combined into the *SCION Control Service*. This consists of the path server and also services to handle incoming PCBs.

2.3 SCIONLab

Having described the SCION protocol and implementation concepts, also existing SCION research networks are shown in this work. One working testbed for SCION is *SCIONLab* [19], a development and research network running SCION for interested users where they can test and deploy their own SCION projects. SCIONLabs core idea is to allow for a quick setup and low entry bar for new users with less technical experience to run a first SCION AS on their own hardware. Each user AS in SCIONLab is attached to a so called *attachment point*. Attachment points provide up-path connectivity to the SCIONLab network. SCIONLab provides pre configured virtual machines and an automated setup for newly created SCION ASes. Furthermore, the SCIONLab website provides information to simplify the onboarding process and the AS configuration. Per default, the user AS only knows the

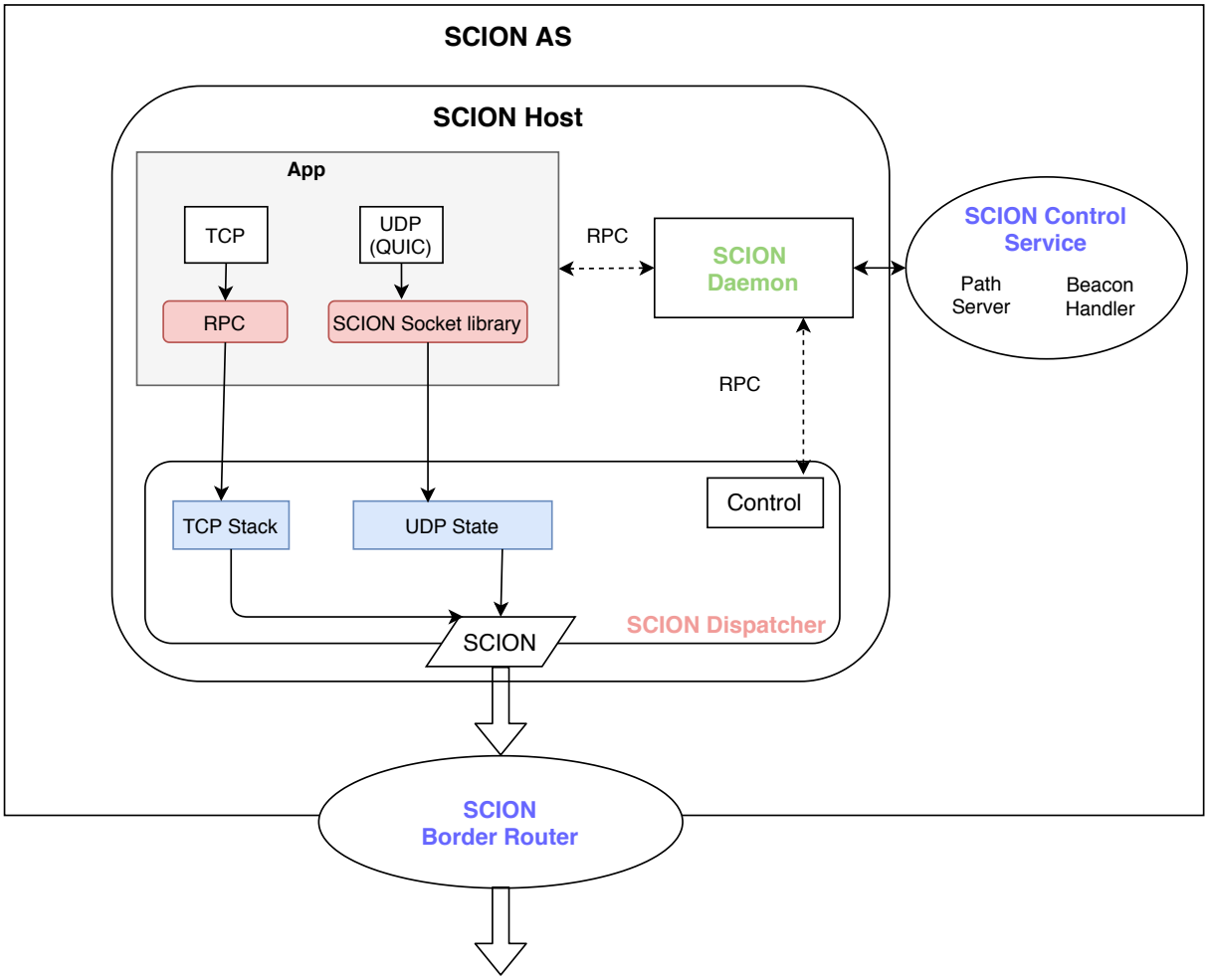

Figure 2.8: Host Architecture of a SCION end Host running also AS services

connection to its attachment point, but through the configurable topology of SCIONLab, new links to other ASes or end hosts can be easily added using configuration files. However, SCIONLab comes with some restrictions compared to a production SCION network. It provides a centralized management of the control plane public key infrastructure, which may be considered as single point of failure. Furthermore, the up-links to the attachment points are overlay links over the publicly routed Internet. Consequently, not all benefits of SCION can be fully realized in this testbed.

3 Multipath Bittorrent over SCION

This section is structured as follows. First, the foundation of this work, the existing Bittorrent implementation is presented, followed by the adaptions that were made to implement SCION as transport protocol of Bittorrent. Furthermore, configurations to evaluate Bittorrent over SCION are shown. Finally, the design of Multipath Bittorrent over SCION discussed.

3.1 The original Bittorrent implementation

Bittorrent over SCION is based on a widely used Bittorrent implementation written in Go [20], called *go-torrent* for the rest of this work. Along with the benefits of Gos lightweight concurrency using go routines, go-torrent already supports multiple transport protocols. At the current state, the Transmission Control Protocol (TCP) and the Micro Transport Protocol (µTP) [21] are implemented. Since Bittorrent over TCP suffers from poor latency and other congestion control problems [22], µTP became a popular and efficient alternative to TCP for Bittorrent.

Go-torrent supports the following storage backends: 1) a *file-backend*, which writes each downloaded piece at the correct position of the resulting file in the file system 2) a *bolt-backend* which used bolt-db [23] as storage for requested pieces and 3) a *mmap-backend* [24] providing (volatile) in-memory storage. Also the measurement of important metrics during the download process is already implemented, e.g. requested, received and duplicated pieces. These metrics are collected per torrent as well as per connection. Go-torrent accepts torrent files as input to start a download process. Furthermore, providing a magnet link is another supported input for go-torrent. Magnet links are resolved to their referenced torrent files which are afterwards downloaded.

The download process of a torrent file is implemented as follows. For each torrent, a set of peers that provide the pieces of the torrent is created. A peer is defined as a single IP address, which may be an IPv4 or IPv6 address. These peers are usually gathered from the tracker or added via the peer exchange protocol. Given a set of peers, figure 3.1 shows go-torrents implementation of the download process.

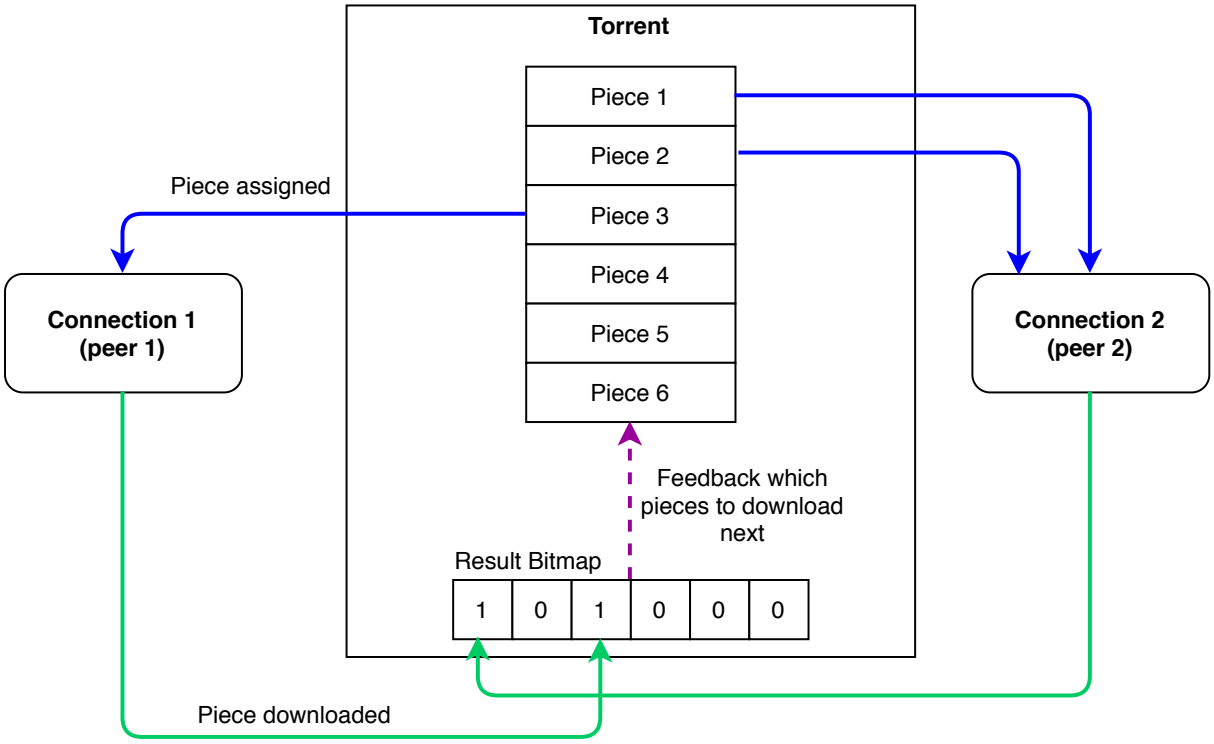

Figure 3.1: Go-torrent implementation of downloading a torrent file over multiple connections.

To begin the download, the following parameters are required: 1) an array of *peers*, 2) the *pieces* to request, and 3) *maxRequests*, the maximum number of pipelined requests. Go-torrent creates a bitmap which stores the state of each *piece*. A *piece* is either *open*, *requested* or *downloaded*. At the beginning, go-torrent opens a connection to each available peer. To each successfully established connection *con*, *maxRequests* pieces are assigned. These *pieces* are requested and downloaded over the connection. The *request strategy* determines the assignment of pieces to connections. Go-torrent supports the following request strategies:

- **Fuzzy prioritized**: Favour higher priority pieces with fuzzing to reduce overlaps and wastage across connections.
- **Prioritize fastest**: Fastest connection is prioritized, then the other connections follow.
- **Strictly prioritized**: Pieces are downloaded strictly by priority and duplicates are avoided.

These request strategies are not exact implementations of the strategies shown in chapter 2. Per default *Prioritize fastest* is enabled. *Strictly prioritized* aims to reduce overall bandwidth by avoiding duplicate piece requests over multiple connections. *Fuzzy prioritized* uses fuzzy logic to assign pieces to connections to reduce piece request overlap.

After performing the piece assignment, the connections send piece request messages to their respective peers and wait until these send back the requested pieces. After each received piece, go-torrent updates the result bitmap and assigns a new, open piece to the connection. The update process of the result map contains more steps than simply changing the value of the received piece. In addition, the integrity of the retrieved piece is verified. For this, the piece is hashed to verify the resulting hash against the one stored in the torrent file for that piece. Upon success, go-torrent writes the retrieved piece to the currently active storage backend.

Go-torrent can be started in different modes, which are configured via command line flags. As one of the most important flags, go-torrent can be started in *seed mode* to allow seeding the torrent without expecting any value back from the downloading peers.

3.2 Current State: Bittorrent over SCION

Based on go-torrent, Koppehel presented an implementation and evaluation of SCION as transport protocol for go-torrent [25]. This work adds QUIC over SCION (SQUIC) as third transport protocol to Bittorrent and a set of configuration flags to enable SCION mode and force go-torrent to use only SQUIC as transport protocol. Along with those extensions, also SCION addresses are supported. However, Bittorrents use of SCIONs path-awareness is only in its infancy. In case of more than one available paths to a SCION peer, per default the first path that is returned from the path query is used.

Bittorrent over SCION introduces support to connect to SCION peers, as well as a new SCION-only mode which forces all connections to be established over the SCION network. As there is no tracker implementation which supports SCION addresses at the time of writing, another way to bootstrap an initial list of peers was needed. This was solved by simply passing a static set of scion peer addresses into the torrent client. The APPQUIC library [26], based on a QUIC implementation [27] in Go is used as simple interface of QUIC over SCION to add connection management and congestion control and provide the necessary transport features.

Initial performance benchmarks of Bittorrent over SCION were already presented [9]. These benchmarks only analyzed Bittorrent over SCION using single paths to each peer. However, one of the core benefits of SCION is its path awareness. Therefore, in this work, Bittorrent over SCION is extended to use multiple paths to a single peer in parallel.

3.3 Designing Multipath Bittorrent over SCION

In some sense Bittorrent already has a built-in multipath approach, which works well in case of many peers [8]. However, when considering a setup of only a few peers that are reachable over multiple paths, the current approach only reaches a fraction of the theoretically achievable bandwidth. This work addresses this problem by switching the peer definition from an address to a combination of address and path.

3.3.1 Path-level granularity

Normally, a Bittorrent peer is represented solely by its address. This address could be either an IPv4, IPv6 or SCION address. Peers that are only represented by their address are called *address-level* peers for the rest of this work. To allow the usage of multiple SCION paths to a particular peer, a new peer representation called *path-level* peer is introduced. Path-level peers are always SCION peers. The representation of a path-level peer is changed from using only the SCION address to a tuple $(addr, path)$, consisting of the SCION address and one possible path to this peer.

Generally, peers that are returned from a tracker or added via static bootstrapping to Bittorrent are always address-level peers. Consequently, they do not contain any path information. Thus, a path lookup is required to determine the path for this peer. To use multiple paths to a single SCION peer, a mapping from address-level peers to path-level peers is performed. Since mostly more than one path is available to an address-level peer, the path-level peer list may contain more entries than the original address-level peer list.

Algorithm 1 shows the approach used to map address-level to path-level peers. The algorithm input is the address-level peer list anda maxConnectionsPerPeer field. This field provides an upper limit of paths to each peer to avoid potential excessive growth of the path-level peer list. For each address-level peer, a SCION path lookup is performed, by calling the *queryPaths* function. SCION returns already sorted path in descending order by the number of hops for a lookup request. Since the SCION path lookup result may contain paths multiple times, any duplicates are removed using *unique*. Afterwads, the number of created path-level peers *numPathLevelPeers* is calculated using the minimum value of the number of returned paths and maxConnectionsPerPeer. In the for loop, numPathLevelPeers are created using the address-level peer and the ith path.

An alternative to the path-level peer approach is to keep the peer definition untouched and implement multipath on socket level, meaning to establish one connection to a particular peer which uses multiple paths under the hood. A drawback of this alternative is the required reimplementation of all the statistical measurement functionality already

```
Data: addrPeerList, maxConnectionsPerPeer
Result: pathPeerList
pathPeerList = [];
forall Peer peer in addrPeerList do
    paths := unique(queryPaths(peer));
    numPathLevelPeers := min(len(paths), maxConnectionsPerPeer);
    for i = 0; i < numPathLevelPeers; i++ do
        pathPeerList.push(peer, paths[i]);
    end
end
```

Algorithm 1: Mapping of address level peers to path-level peers

supported by go-torrent. Furthermore, implementing multipath on socket level infers to design a new approach to assign pieces to particular paths of connections instead of using Bittorrents thoroughly researched multipath design.

Per default, Bittorrent over SCION uses up to a fixed number of paths. In case this number is selected large enough, all available paths to each peer are used. Naturally, not all available paths provide a similar bandwidth. Considering a diverse setup of high and low bandwidth paths, using all paths may not be the best option. Consequently, this work uses built-in SCION functions to determine which paths to use. The main benefit of this approach is that no prior knowledge of the network topology, especially the number of available paths to each peer, is required.

3.3.2 Path Selection

Multipath support on network level imposes the problem of finding an optimal set of paths under the current conditions. What is optimal depends on the point of view: For a client accessing resources, performance maximization is one possible optimization criterium. As network operator, traffic engineering factors like bandwidth concentration or resource balancing are possible optimization goals.

For all of these goals, up to date information about paths are necessary. SCION per default provides two additional information fields for each path: 1) a bandwidth field contains the maximum available bandwidth of the underlying link and 2) the configured MTU. These static information can be used to pre-sort of the paths, but dynamic information collection about paths is necessary to fully utilize the multipath capabilities of SCION.

One core challenge of finding the optimal path set are changing properties. Paths can be congested or may even fail during the download process. A good path chosen at the beginning may become undesirable later. In this work, the download process is divided into time slots. A path set is always considered at a specific time slot. In theory, the

smaller the time slots are, the more accurate becomes the possible adjustment of the optimal path set. In practice, smaller time slots lead to higher computing effort for finding optimal paths sets. Furthermore, smaller time slots may more likely lead to more path switches, because less information is available.

In the scope of this work, download performance maximization and avoidance of congested paths are the optimization goals for path sets. To determine the quality of a path set at a given time slot, multipath Bittorrent over SCION implements a *connection racing* approach to weigh paths. In the scope of this work, a path is defined as congested in case its weight is under a specific percentage of the best paths weight. The path weighing takes place according to the phases of the download process shown in Figure 2.1. Before the actual download starts, a connection handshake is required. In the first phase, the amount of time required to handshake a connection over a specific path is used as path weight. The longer a handshake over a connection took, the lower is the weight of the respective path. Determining the set of paths according to their handshake time is called *initial path selection*. In the download phase, bandwidth statistics are collected for each path. Therefor, a path is weighted according to its download bandwidth. The path selection in this download phase is called *intermediate path selection*.

Initial path selection

The measurement of the handshake duration is already implemented in go-torrent. The initial path selection is performed either after all connections completed their handshakes or after a predefined amount of time. Algorithm 2 shows the design of the initial path selection process. The algorithm input is a list of handshook connections and a field determining the minimum percentage of path weight compared to the best path, called *minWeightPercentage*. Since each connection uses exactly one SCION path, this connection list can be considered equal to a path list. Multipath Bittorrent over SCION selects a path in case its weight is greater or equal compared to *minWeightPercentage* percent of the best path. At first, all connections are sorted by the duration of their handshake time in ascending order. Next, the weight calculation of the best connection is performed, the result is called *bestWeight*. For each connection the weight is calculated depending on the respective handshake time. This weight is compared to *bestWeight* with respect to the *minWeightPercentage* passed to this algorithm. The connection, and respectively the path, is selected if the weight reaches at least a percentage of *minWeightPercentage* of the *bestWeight*. After the iteration over all connections is finished, the unselected connections

are closed and no longer used to download.

```
Data: connectionList, minWeightPercentage
Result: selectedConnectionList
selectedConnectionList = [];
sortedConnectionList = sortByHandshakeTime(connectionList);
bestWeight := weightHandshakeTime(sortedConnectionList[0]);
forall Connection con in sortedConnectionList do
    weight := weightHandshakeTime(con);
    if hasMinimalWeight(bestWeight, weight, minWeightPercentage) then
        selectedConnectionList.push(con);
    end
end
```

Algorithm 2: Initial path selection algorithm

The design of the initial path selection approach imposes advantages as well as drawbacks. The first advantage is that a path selection can be performed without the needing to download any data. Consequently, it is a fast way to perform a path selection at the beginning of the download process. Considering the intended fairness of Bittorrent, no upload bandwidth has to be returned to other peers with this approach. Nevertheless, using the handshake time to optimize download bandwidth is an inaccurate approach. The handshake duration depends on the latency of the link, since a low amount of data is transferred.

Intermediate path selection

In contrast to the one time selection of the initial paths, the intermediate path selection is performed multiple times over the download process. Consequently, this approach requires an additional variable: the *timeSlot*. TimeSlots are defined in milliseconds. At every timeSlot the path selection algorithm determines the selected path set for the period until the next time slot is elapsed. As information to weight paths, the download bandwidth within the period until the next time slot is used. Figure 3.2 shows the process of tracking the bandwidth over time and performing the path selection using the measured data. Go-torrent tracks the absolute number of read and written bytes for each connection. This field is updated each time a piece was retrieved successfully over any connection. The bandwidth tracking designed in this work adds a map to each connection, which stores the bytes read within a time period. A timer is used to trigger a path selection event after each elapsed timeSlot. This event is handles in two steps. At first, the number of read bytes for the last time period is calculated. For timeSlot t, this is done by subtracting the number of read bytes of timeSlot t-1 from the overall number of read bytes of the

connection. Secondly, after all connections performed this step, the intermediate path selection algorithm performs the selection for timeSlot t.

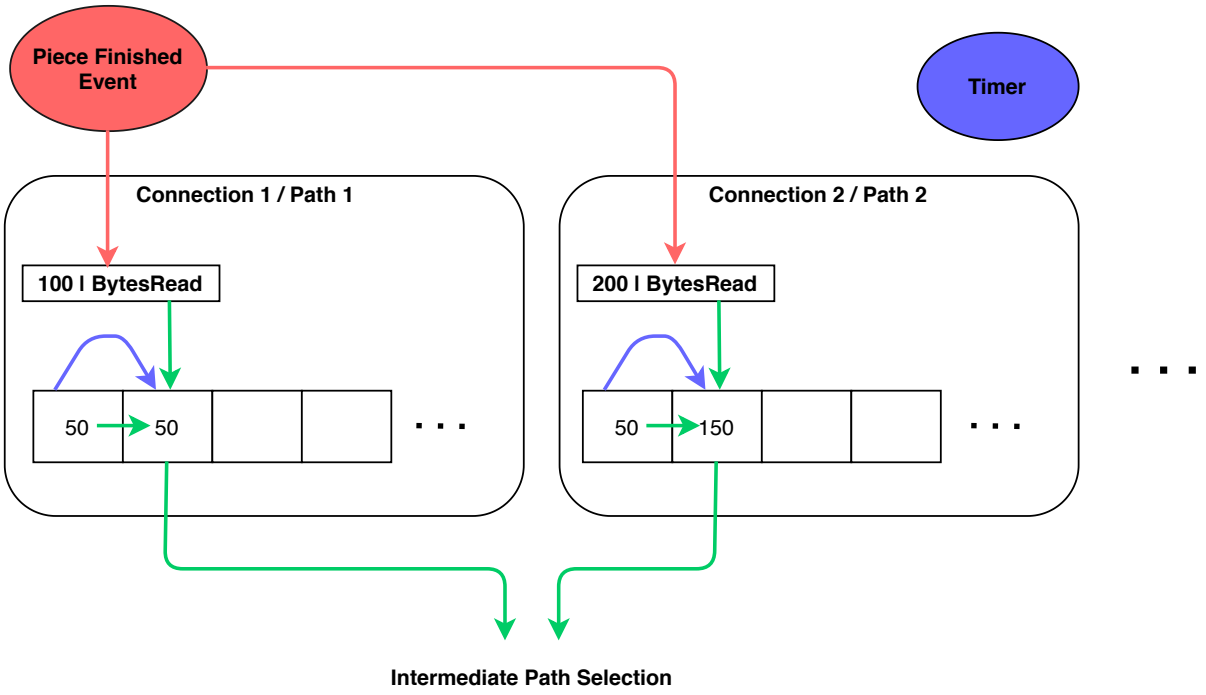

Figure 3.2: Path level bandwidth tracking and intermediate path selection design

Algorithm 3 shows the implementation of the intermediate path selection. At first, the connections are sorted in descending order by the number of bytes read in the current timeSlot. Secondly, the *bestWeight* is calculated from the first connection of the newly sorted array. This is done by using the number of read bytes. As next step, all the algorithm iterates over all connections and calculates the weight for each connection. The algorithm compares this weight to the previously calculated *bestWeight*. To compare the weights, again the *minWeightPercentage* is used, similar to the initial path selection.

Afterwards, all unselected connections are closed.

```
Data: connectionList, minWeightPercentage, timeSlot
Result: selectedConnectionList
selectedConnectionList = [];
sortedConnectionList = sortByLastBandwidth(connectionList, timeSlot);
bestWeight := weightLastBandwidth(sortedConnectionList[0], timeSlot);
forall Connection con in sortedConnectionList do
    weight := weightLastBandwidth(con, timeSlot);
    if hasMinimalWeight(bestWeight, weight, minWeightPercentage) then
        selectedConnectionList.push(con);
    end
end
```

Algorithm 3: Intermediate Path selection

Similar to the initial path selection, the intermediate path selection has advantages and drawbacks. The first main advantage of this approach is the accurate weighting of paths using the achieved bandwidth. Furthermore, this approach does not require additional data to be transferred, since it uses the anyway downloaded data by Bittorrent. A possible drawback of this approach is the need of unchoked connections over the respective paths to weigh them accurately.

4 Benchmark Setup

In order to evaluate the multipath design and implementation of this work, a comparison of the SCION candidates among each other is planned as well as comparing Multipath Bittorrent over SCION to a path-unaware approach.

4.1 Network Topology

To provide a reliable test setup for multipath Bittorrent over SCION, a setup of multiple ASes connected via high performance links is used. Figure 4.1 shows the used network topology for the benchmarks performed in this work. The selected topology consists of of two servers connected via two links, one 1Gbit/s link through a switch and one 10Gbit/s link directly connecting both servers. Each server has a 10 core Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz and 48 gigabytes of main memory. The used network cards are Intel X722 NICs. Each server runs Ubuntu 18.04 with a linux kernel 4.15.117. This setup is used to evaluate multipath Bittorrent over SCION for several reasons. At first the servers provide high performance CPU and network cards to allow multi-gigabit bandwidths. Secondly, having one 1Gbit/s and one 10Gbit/s link, an evaluation over significantly different links regarding performance is possible. Furthermore, with a 10Gbit/s link, possible limitations of Bittorrent regarding the maximum available bandwidth on single links can be evaluated. Consequently, this topology matches the targeted use case scenario of having a low number of fast connected peers.

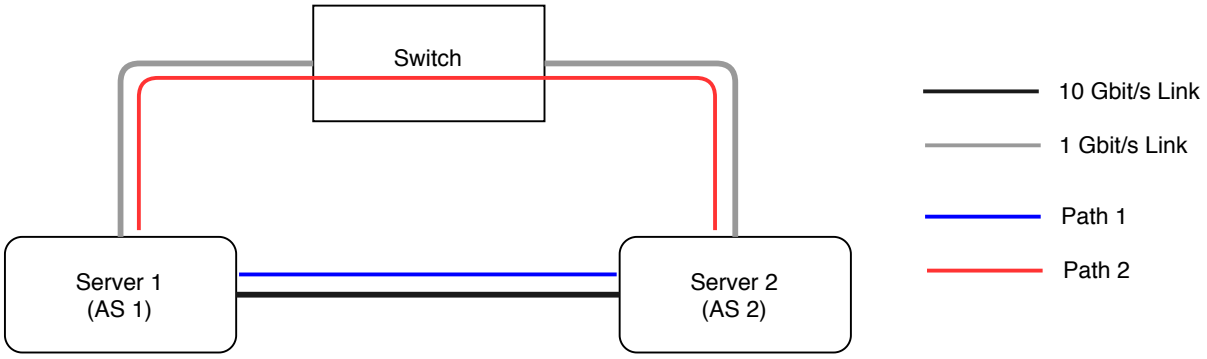

Figure 4.1: Benchmark setup of two servers that run 2 SCION ASes connected via one 1Gbit/s link and one 10Gbit/s link.

4.2 Bittorrents Fairness

Fairness is an important concept in Bittorrent to reach equal upload and download bandwidth for each peer. This is done by the presented choking algorithms. Bittorrent begins choking after a fixed number of peers is requesting data. Originally, only 4 unchoked peers are connected to each seeder. Within the presented benchmark setup, no choking is performed, because only two path-level peers are downloading from the seeder. However, in larger setups consisting of a huge number of path-level peers, choking may significantly impact download performance of particular peers.This probably happens if connections over fast paths are choked whereas slow paths are unchoked. With the presented path selection algorithms, Bittorrent is capable of measure path performance to probably avoid slow paths being unchoked.

4.3 Benchmark candidates

The following variants of Bittorrent over SCION are analyzed in this work.

4.3.1 Singlepath Bittorrent

The first benchmark candidate used in this work is the existing Bittorrent over SCION implementation. For all following benchmarks, this candidate is called *SCION Singlepath*. The implementation of this candidate was left unchanged. Consequently, only its only capable of using one SCION path to a single peer.

4.3.2 Singlepath Bittorrent with HSR

From previous tests with SCION, an assumed performance limitation of the open source border router [28] arises. Per default, this router is implemented in the SCION stack shown in Figure 2.8. In case of using an alternative router leading to significantly higher performance, the assumption of this bottleneck can be verified. From anapaya systems [29], a high performance SCION border router was provided for research projects based on SCION, called *HSR*. The HSR tackles the problem of handling a huge number of packets in user space by using the DPDK [30, 31] plugin of VPP [32]. VPP is a framework providing a high performance network stack that runs on the CPU. DPDK is set of libraries and polling-mode drivers to handle packets in user space. The main benefit of this approach is the kernel bypass of packets to avoid slow packet transfer into user space. Figure 4.2 shows the mechanics of DPDK. The core mechanism is to split a NIC into physical functions

and virtual functions. The physical functions remains in control of the linux kernel and is accessible as network interface for programs. From the linux kernel, filters can be programmed that pass incoming packets to the matching function. Programs developed with DPDK can directly access virtual functions without using the way over the linux driver loaded to communicate with the physical functions. This avoids loading incoming packets first into the kernel before moving them into user space. Furthermore, packet handling via kernel interrupts does not reach sufficient performance for particular use cases. DPDK uses polling to retrieve packets to avoid this limitation.

The HSR is available as docker container that runs in privileged mode to gain full access to the network interfaces. To configure the HSR, SCION uses the beforehand created virtual functions on both interfaces as ingress and egress interfaces. This benchmark candidate is called *SCION Singlepath HSR* for the rest of this work.

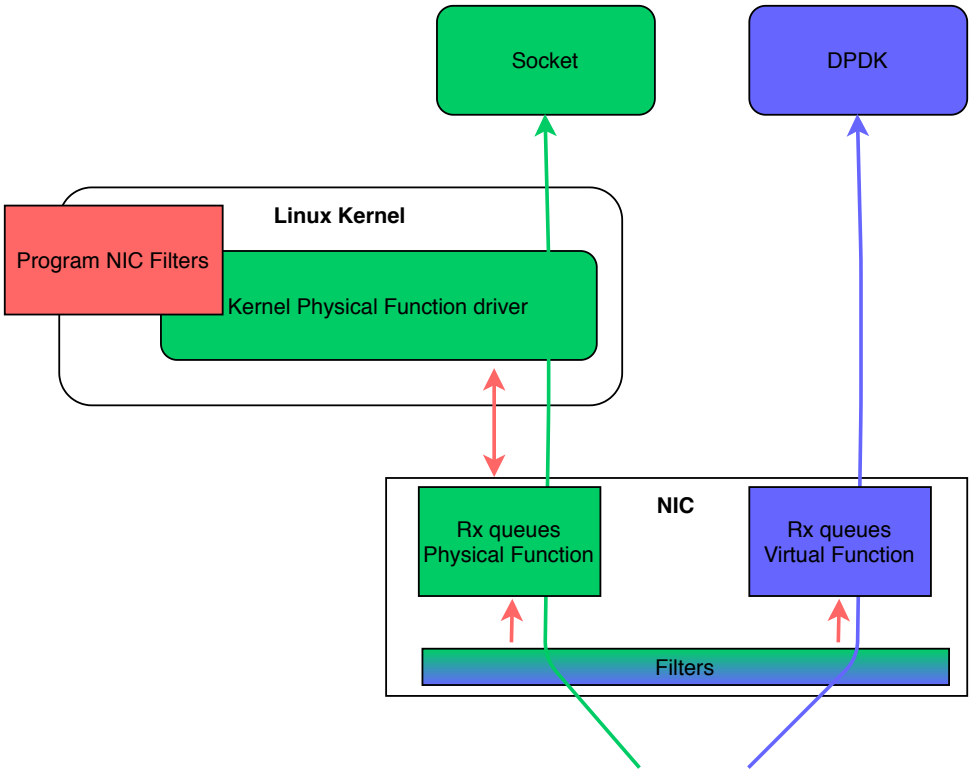

Figure 4.2: DPDK kernel bypass mechanics, adapted from [33]

4.3.3 Singlepath Bittorrent with Jumbo Frames

As the assumed performance limitation of the open source border router is based on the number of handled packets, this approach enables jumbo frames to be used by Bittorrent over SCION. Jumbo frames allow sending packets with up to 9000 bytes MTU for the used network cards. To enable jumbo frames, the MTU of the used ingress and egress interfaces is set to 9000. Furthermore, the respective MTU is set in the SCION interface

configurations. This approach bases on improving the performance of the open source border router, since the HSR does not support jumbo frames at the current state. To enable sending of jumbo frames via QUIC, the used quic-go implementation was adapted by increasing the maximum packet size to 9000. The variant of Bittorrent over SCION that uses a single path with jumbo frames is called *SCION Singlepath JF*

4.3.4 Multipath Bittorrent

The first multipath benchmark candidate is called *SCION Multipath*. It extends the SCION Singlepath candidate by aggregating bandwidth of multiple paths. This candidate uses the open source border router and the default MTU of 1400 byte.

4.3.5 Multipath Bittorrent with HSR

This benchmark candidate called *SCION Multipath HSR* extends SCION Singlepath HSR by aggregating bandwidth over multiple paths. Since the HSR does not support jumbo frames in the used version, again the default MTU of 1400 byte is used.

4.3.6 Multipath Bittorrent with Jumbo Frames

The last presented benchmark candidate is called *SCION Multipath JF*. It uses the designed multipath implementation running the open source border router with jumbo frames enabled.

5 Evaluation

The evaluation presented in this work is split into 4 sections. At first, the existing Bittorrent measurements performed by Koppehel [9] are summarized. Next, all presented SCION benchmark candidates are analyzed. Afterwards, SCION is compared to a path-unaware approach based on UDP. Finally, the performance of Bittorrent over SCION is evaluated using a varying number of torrents as well as different file sizes.

5.1 Existing measurements

As first part of the evaluation, results of the original Bittorrent over SCION implementation are summarized. In this work, Koppehel present different experiments to evaluate Bittorrent over SCION. This work focuses on two of them: The performance benchmark on a 1GBit/s link and the multipath example.

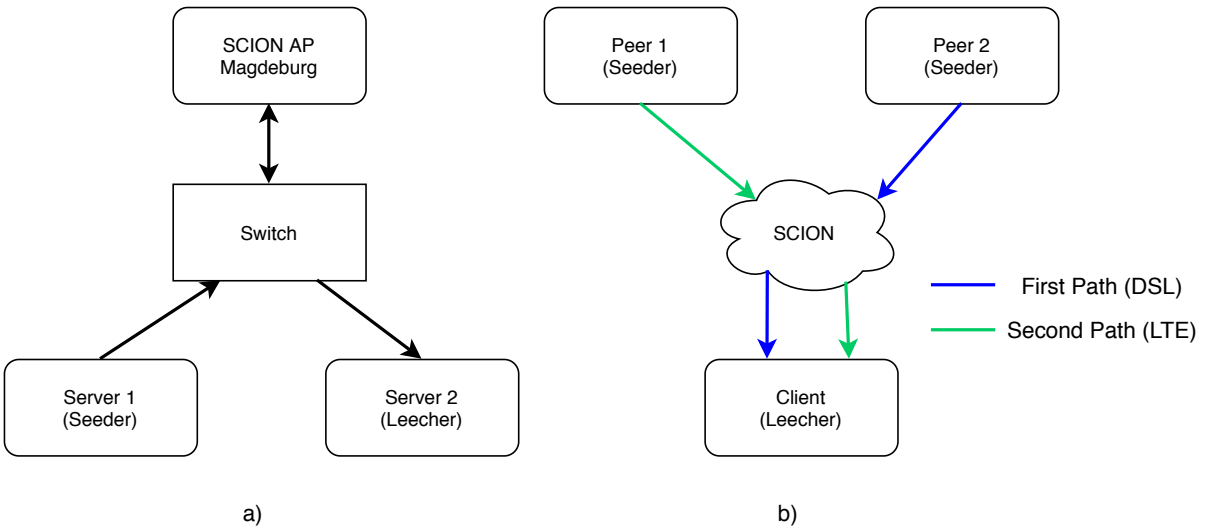

Figure 5.1: Existing measurements of bittorrent over SCION consisting of a) performance benchmark and b) multipath benchmark, adapted from [9]

Figure 5.1a visualizes the setup for the performance of the initial Bittorrent over SCION implementation [9]. For this experiment, the same two servers are used that were mentioned in the previous chapter. Furthermore, a seeder of Bittorrent over SCION is deployed on Server 1 and a leecher on Server 2. Both servers and the attachment point are each connected via 1 Gbit/s links through a switch. Since the traffic has to flow to and from

the SCION AP over the same link, the achievable bandwidth is halved. Bittorrent over SCION results in 487 Mbit/s, which is close to the expected 500 Mbit/s.

Figure 5.1b shows the multipath example presented in [9]. This setup consists of two remote peers Peer 1 and Peer 2. Both are connected over a 100 Mbit/s DSL line and a LTE line providing a maximum of 50 Mbit/s. Without the multipath feature of SCION, a classic setup of Bittorrent uses only the DSL line and reaches in total 95 Mbit/s. In contrast, Bittorrent over SCION uses both paths together, reaching 43Mbit/s over LTE and 94 Mbit/s over DSL. Aggregating both paths results in total a bandwidth of 137 Mbit/s. Consequently, using the multipath feature of SCION, a bandwidth increase of 44% is possible in this setup.

5.2 SCION candidates

After presenting the existing measurements for Bittorrent over SCION, the benchmark candidates presented in chapter 4 are analyzed. This experiment is performed using the evaluation setup presented beforehand in Figure 4.1, consisting of two ASes connected with a direct 10Gbit/s link and a 1Gbit/s link over a switch. As default request strategy, *Prioritize Fastest* is used. The singlepath benchmark candidates all use the 10Gbit/s path, whereas the multipath candidates use both paths in parallel.

At first, the time required to download a 5 Gigabyte file is measured for all candidates. Afterwards, the actual download bandwidth is analyzed and the CPU usage is evaluated, downloading the same file. Finally, the CPU usage is analyzed for all participating SCION components.

This experiment aims to compare the presented benchmark candidates to find the best performing one. Furthermore, a suspected bottleneck in the open source border router is investigated through a comparison with a setup using the HSR. The performance increase of using the HSR as well as jumbo frames compared to an unchanged SCION setup is evaluated. Finally, the performance increase of aggregating multiple paths is compared to a singlepath approach.

By aggregating the bandwidth of 2 paths compared to using only one path, up to a 2 fold increase in performance is expected for the multipath candidates compared to their singlepath counterparts. Due to the suspected performance bottleneck in the open source border router, the HSR setup is expected to clearly outperform the open source variant, for the singlepath and multipath candidates. Furthermore, the usage of jumbo frames to overcome a limitation on the number of processed packets is expected to reach up to 6 times performance increase compared to using the default MTU of 1400.

5.2.1 Download Time

Figure 5.1 shows the average time required to download a 5 Gigabyte file by all candidates. The process is repeated 5 times to achieve meaningful results. The standard deviation of all repetitions for each candidate is also presented.

	Average download time [s]	Standard deviation [s]
SCION Singlepath	107.2	0.75
SCION Singlepath HSR	96.2	0.4
SCION Singlepath JF	43.8	0.4
SCION Multipath	82	0
SCION Multipath HSR	65	1.2
SCION Multipath JF	32.3	0.45

Table 5.1: Download time and its standard deviation in seconds of a 5 Gigabyte file downloaded by all SCION candidates 5 times.

The best performing candidate is SCION Multipath JF with an average download time of 32.3 seconds, followed by SCION Singlepath JF with 43.8 seconds. The HSR candidates reach average download times of 96.2 and 65 seconds for singlepath and multipath, respectively. The highest download time is observed by SCION Singlepath with 107.2 after SCION Multipath with 82 seconds. Finally, the standard deviation of all candidates is lower than 1.2 seconds.

Enabling jumbo frames lead to a significant performance increase for singlepath and multipath candidates. Nevertheless, only around 2-3 times performance increase is observed in contrast to the expected 6 times. Also the performance increase of 2 times fold by aggregating multiple paths is significantly lower than expected. The HSR candidates does not deliver a high performance increase compared to the traditional SCION candidates. Consequently, this indicates another possible performance bottleneck in addition to the one suspected in the open source border router.

5.2.2 Download Bandwidth

To more precisely evaluate the performance, the actual bandwidth over the download process is measured in the next step. This measurement is performed with the same setup used to evaluate the download time. Since the download time differs significantly, the bandwidth of all candidates is measured at intervals during the completion of the download. This allows a comparable presentation of all candidates. Figure 5.2 shows the comparison of the average download bandwidth of all benchmark candidates. Again, the

measurements are repeated 5 times. The standard deviation is presented as error bars around the data points.

Based on the previously achieved download times, SCION Multipath JF and SCION Singlepath JF are expected to reach the highest bandwidth, followed by SCION Multipath HSR and SCION Multipath. Finally, SCION Singlepath HSR is expected to be second to the last, SCION Singlepath. Although the previously measured download time allows indications about the achieved bandwidth, no propositions about the development over time can be concluded.

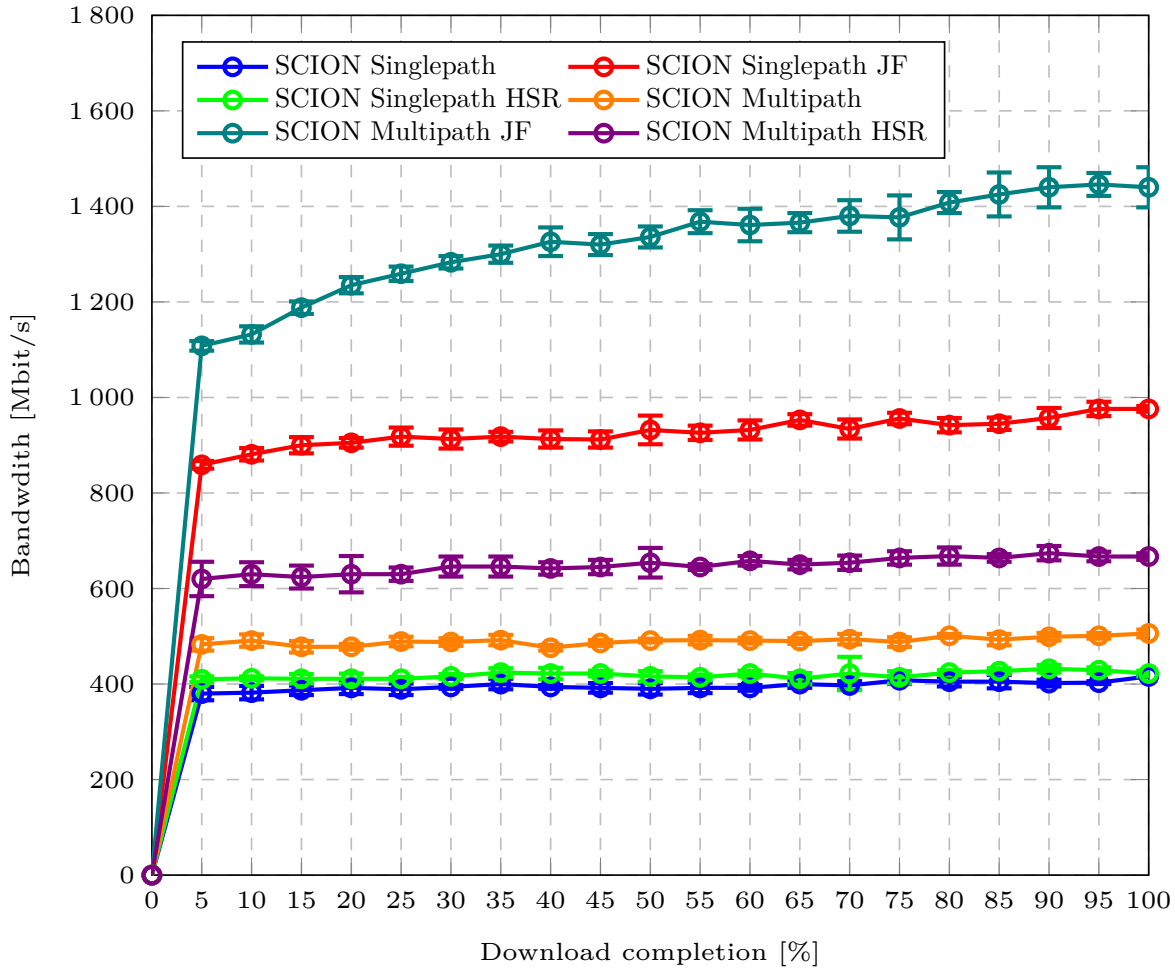

Figure 5.2: Comparison of the average download bandwidth of all benchmark candidates downloading a 5 Gigabyte file with 5 repetitions expanded by the resulting standard deviation.

According to the download time measurements, the jumbo frames candidates reach the highest bandwidth, with around 1400 Mbit/s and up to 1000 Mbit/s for SCION Multipath JF and SCION Singlepath JF, respectively. The SCION Multipath HSR reaches slightly over 600 Mbit/s compared to the traditional SCION Multipath with 500 Mbit/s. At around 400 Mbit/s, SCION Singlepath and SCION Singlepath HSR are located. The candidates that use jumbo frames show a bandwidth increase to the end of the download, especially

SCION Multipath JF. However, the HSR and traditional SCION candidates deliver a constant bandwidth over the complete download. Furthermore, they reach their maximum bandwidth already at 5% of the download process, in contrast to the candidates using jumbo frames. Except for SCION Multipath JF, the deviations in the measurements are comparatively small. The higher standard deviation of SCION Multipath JF is suspected to be an artifact of the higher achieved bandwidth.

The bandwidth increase over the download process can be explained by the leverage of jumbo frames to the quic-go and Bittorrent implementations. In contrast to the expected performance increase by replacing the open source router with the HSR, only slight improvements are observed. By using one path, the HSR is nearly head to head with the traditional singlepath approach. Compared to SCION Multipath, a performance increase of around 20% is reached by the HSR using both paths.

Although not all expected performance improvements can be observed, at least the jumbo frames candidates deliver significant improvements. A possible explanation for the shortfall in performance could be a bottleneck somewhere earlier in the SCION software stack that limits the amount of packets that can be processed each second. To investigate this further, the CPU usage of the candidates is analyzed next.

5.2.3 Overall CPU Usage

Along with the actual download bandwidth, the CPU usage of the candidates promises helpful information to correlate with their performance. Therefor, the next experiment step is the evaluation of the aggregated CPU usage of all involved SCION components of the candidates. Since no candidate entirely uses the available bandwidth, they are expected to be CPU-bound in some form.

Figure 5.3 shows the aggregated CPU usage of all SCION components over the complete download process. In contrast to the expected CPU limitation, all candidates consume between 40% and 65% overall CPU usage. Therefore, surprisingly, the reason for not fully using the available bandwidth does not appear to be limited CPU power. Although reaching the lowest download bandwidth, SCION Singlepath does not result in the lowest CPU usage, which is observed for SCION Singlepath JF. With the usage of jumbo frames, less packets have to be processed, which leads to lower CPU usage. However, the amount of data stored in each packet does not leverage the CPU usage. SCION Multipath JF also requires significantly less CPU power compared to the other multipath candidates.

With around 60%, the highest amount CPU power is required by SCION Multipath and SCION Multipath HSR. The HSR candidate reaches around 20% more bandwidth while consuming the same amount of CPU usage. This correlates with the higher number of

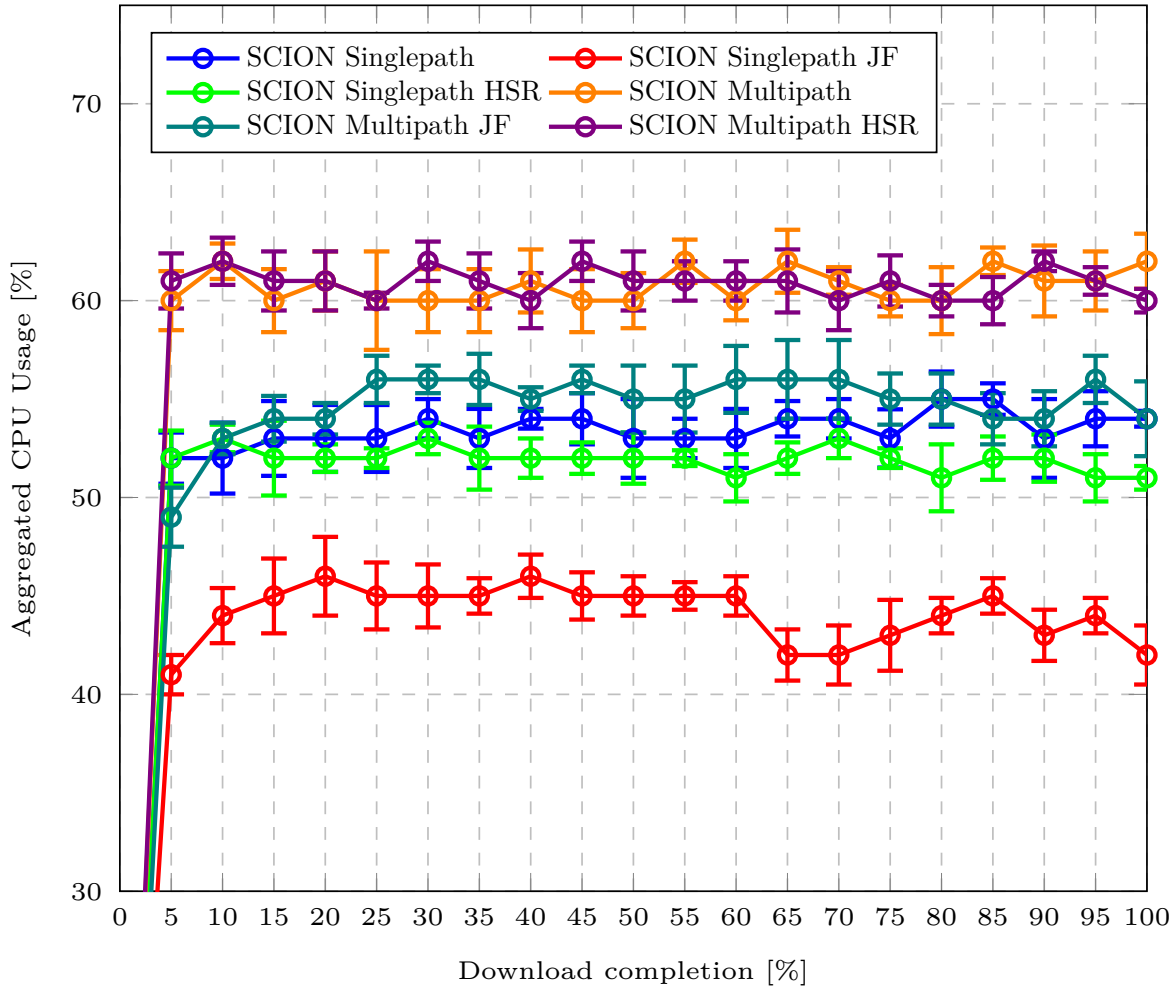

Figure 5.3: Comparison of the average CPU usage of all benchmark candidates downloading a 5 Gigabyte torrent file with 5 repetitions expanded by the resulting standard deviation..

packets that the HSR is available to process. However, the HSR is limited to fully use 1 CPU core at 100% due to its architecture. Having 10 cores, using 1 core at 100% results in 10% overall CPU usage. Consequently, the open source border router is assumed to require more than 10% CPU usage for processing less packets compared to the HSR. Thus, a more detailed view on the CPU usage of the SCION components is presented next.

5.2.4 CPU Usage of SCION Components

To determine the likely candidates for performance bottlenecks in the SCION stack, the dedicated CPU usage of each involved SCION component is measured. The set of considered SCION components consists of the SCION border router (either the open source border router or the HSR), the SCION dispatcher and the Bittorrent application using the SCION socket library to communicate with the SCION dispatcher.

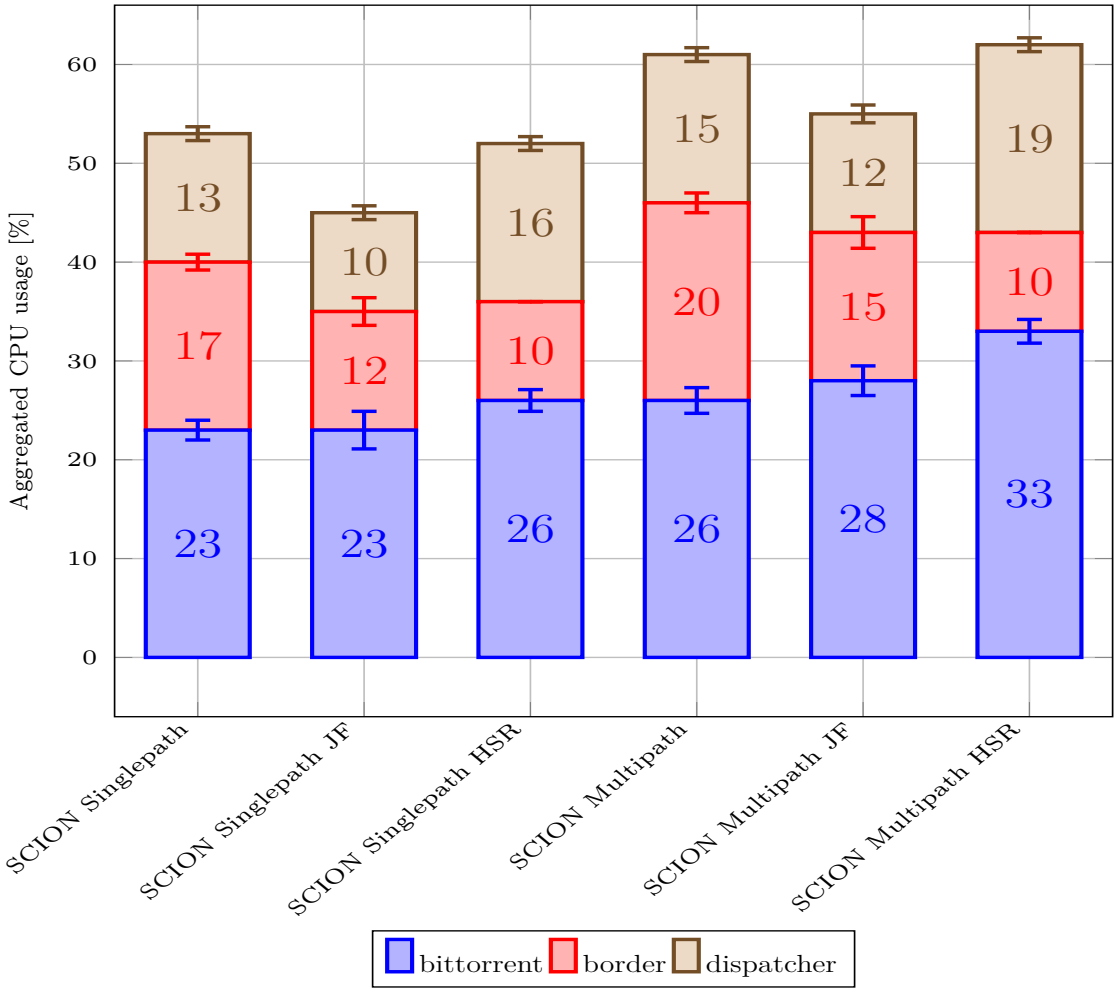

Figure 5.4: Multipath bandwidth aggregation CPU usage of SCION components compound of Bittorrent, the SCION border router and the dispatcher.

Figure 5.4 shows the average CPU usage of each SCION component for each benchmark candidate. Covering also the usage of the SCION socket library, Bittorrent uses the largest amount of CPU power compared to the remaining SCION components. Due to its polling architecture, the HSR uses in both candidates exactly one core, resulting in 10% CPU usage. However, the other components of the HSR candidates require higher CPU power compared to the remaining ones. According to the aggregated CPU usage, SCION Multipath JF and SCION Singlepath JF consume less CPU usage than the other candidates in relation to the achieved bandwidth. Also the dispatcher and border router CPU usage is not significantly impacted by the packet size. The higher amount of CPU power consumed by Bittorrent for the jumbo frames candidates is caused by the required computation power to process the higher number of incoming pieces per second. The HSR candidates consume the highest amount of CPU power in relation to their achieved bandwidth. This is caused by the higher number of packets that are processed, especially for SCION Multipath HSR. This processing effort is reflected in the dispatcher, which is suspected to contain an additional performance bottleneck.

5.2.5 Summary

In this experiment, the presented SCION candidates were evaluated downloading a single file. The download time, achieved bandwidth and the CPU usage was analyzed. It was observed, that neither the jumbo frames nor the HSR candidates deliver the initially expected performance improvements. Furthermore, the bandwidth aggregation of using both paths together remained significantly below the expected increase of factor 2. While evaluating the overall CPU usage, it was observed that increasing the MTU does not lead to significantly higher CPU utilization. However, by analyzing the CPU usage of the HSR candidates, it was concluded that a higher number of packets elevates the CPU utilization of Bittorrent and the SCION dispatcher. The results hint an additional bottleneck in the SCION dispatcher, which is probably responsible for the difference between the measured and expected results. Despite being under the expectations, adding multipath support leads to an improvement of 25% for the traditional candidates, 40% with jumbo frame usage and 50% by running the HSR.

Until now only SCION candidates were analyzed, with SCION Multipath JF performing best. Next, a comparison between SCION and a path-unaware setup is done.

5.3 SCION QUIC vs Native QUIC

After analyzing the Bittorrent over SCION, the best performing SCION candidate is compared to a path-unaware approach called *Native QUIC*. This approach is based on a UDP connection over IP. Since Bittorrent over SCION uses QUIC as transport protocol, also the path-unaware approach is implemented with QUIC on top of UDP. To prevent packet processing bottlenecks from impacting the measurements, the available bandwidth is limited. Thereby, both approaches should be network bound with no impact from different processing requirements on performance. Figure 5.5 shows the used benchmark setup for this experiment. Again, two servers are used that each run one AS. The 10 Gbit/s link was throttled down to 1 Gbit/s. Furthermore, an intermediate virtual node Server 3 was injected, which offers two 1 Gbit/s paths to Server 1 (one path via the switch) and one to Server 2. Since no additional physical server was available, this node is simulated by limiting the overall download bandwidth of Server 2.

Along with the reduced bandwidth for a fair comparison, additional traffic is sent over one or both links to evaluate how the different approaches react to link congestion. This experiment comes in two variants: 1) One path is congested and 2) both paths are congested with traffic. The aim of sending additional traffic over the links is to evaluate how both approaches react to it. Since Native QUIC uses only one path, the bandwidth is expected

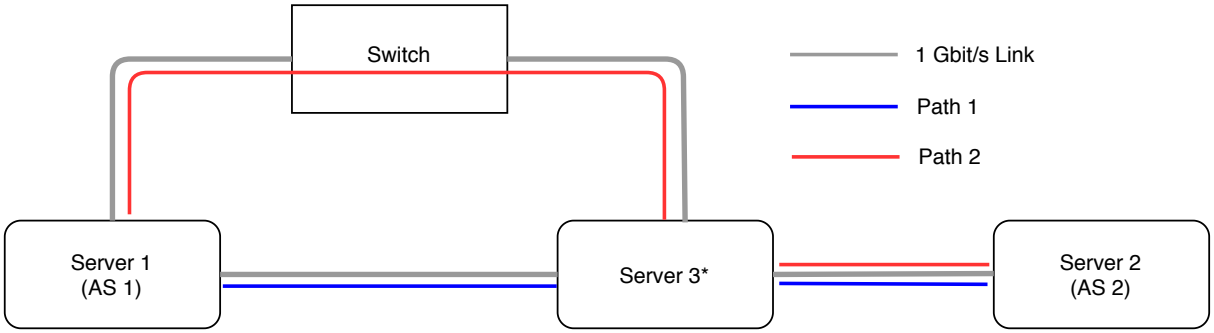

Figure 5.5: Benchmark setup of two servers that run 2 SCION ASes connected via two 1Gbit/s and an intermediate node limiting the bandwidth between the ASes to 1Gbit/s. *The limit was enforced in software.

to drop. In contrast, SCION QUIC is expected to shift the download bandwidth to the free path, resulting in a better performance with constant bandwidth. With congestion on both paths, SCION QUIC should be able to aggregate the remaining bandwidth to outperform Native QUIC.

5.3.1 Congest Path 1

In the first variant of this experiment, Path 1 of the presented benchmark setup is congested. The congestion event is limited to a fixed time window in the middle of each download process. This ensures that both approaches have time to ramp up to the available 1 Gbit/s before and after the congestion event. The congestion events start after 10 seconds of the download process and last for 10 seconds each.

Table 5.2 shows the average time and its standard deviation both candidates require to download a 5 Gigabyte file.

	Average download time [s]	Standard deviation[s]
SCION QUIC	43.6	0.5
Native QUIC	51.9	0.4

Table 5.2: Download time and its standard deviation in seconds of a 5 Gigabyte file downloaded by SCION QUIC and Native QUIC with 10 seconds of congestion on Path 1.

SCION QUIC is with an average of 43.6 seconds compared to 51.9 for Native QUIC around 8 seconds faster. These results indicate that SCION is able to utilize bandwidth from Path 2 during the congestion event.

To support this conclusion, Figure 5.6 shows the actual download bandwidth of both approaches. For SCION QUIC both paths are presented. This figure is truncated on the x

axis to concentrate on the part of the download process where the congestion occurs. No significant changes are observed afterwards.

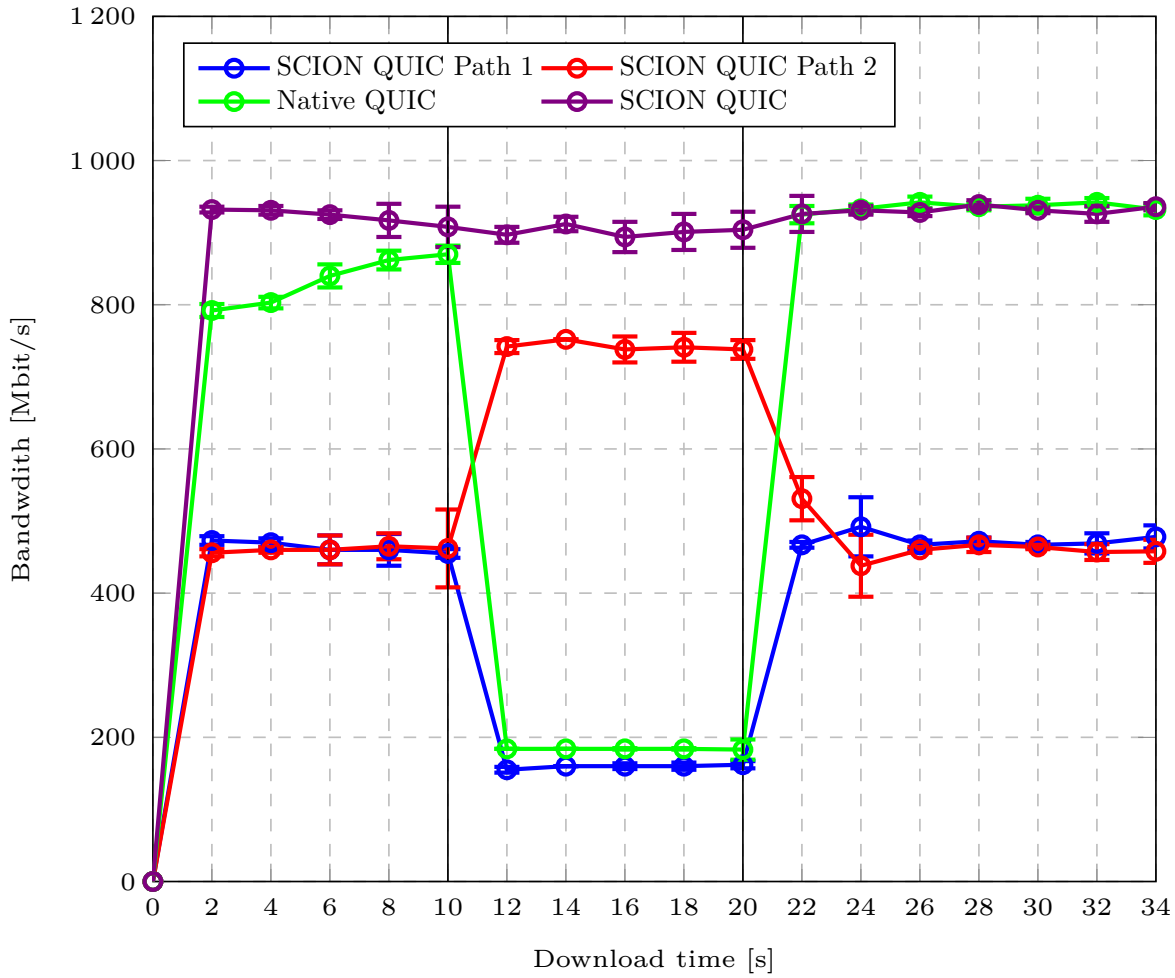

Figure 5.6: Comparison of the download bandwidth of SCION Multipath JF and Native QUIC downloading a 5 Gigabyte torrent file with 5 repetitions expanded by the resulting standard deviation. Between second 10 and 20 Path 1 is congested with traffic, indicated by the vertical black lines.

In the first 10 seconds of the download process, SCION QUIC and Native QUIC nearly reach the available bandwidth of 1 Gbit/s with SCION QUIC performing a faster start. Immediately after the beginning of the congestion event, the bandwidth of Native QUIC drops to around 180 Mbit/s. SCION QUIC on Path 1 also drops down to 150 Mbit/s. Since the congestion control of Native QUIC is more aggressive than that of SCION QUIC on Path 1, it can achieve marginally more bandwidth. While the achieved bandwidth for SCION QUIC on Path 1 drops expectedly, Path 2 is used more heavily and reaches up to 750 Mbit/s. Aggregating both paths, SCION QUIC nearly achieves a constant bandwidth despite the congestion on one path. After the congestion event ends, both approaches quickly restore their original bandwidths of nearly 1 Gbit/s.

As expected, SCION QUIC delivers a nearly constant performance despite one path is

congested. In the next variant of this experiment, both paths are congested with traffic.

5.3.2 Congest Both Paths

This variant of the experiment is designed to determine whether SCION QUIC also outperforms Native QUIC in case traffic is sent over both paths. Again, the same benchmark setup and congestion window are chosen. Native QUIC is expected to perform exactly as in the first variant, because it does not use Path 2. This time however, SCION QUIC is expected to also drop bandwidth, but still achieve around 2 times better performance compared to Native QUIC. Table 5.3 shows the average download time having both paths congested. Compared to Native QUIC, only a small performance increase of around 1 second is achieved by SCION QUIC. This indicates that SCION QUIC may manage to utilize at least some of the bandwidth remaining on both paths.

	Average download time [s]	Standard deviation[s]
SCION QUIC	50.6	1.0
Native QUIC	51.88	0.4

Table 5.3: Download time and its standard deviation in seconds of a 5 Gigabyte file downloaded by SCION and native QUIC with 10 seconds of congestion on both available paths.

To support these initial results, the actual bandwidth is presented in Figure 5.7. Again, the figure is truncated to concentrate on the relevant part of the download process.

Before the congestion starts, both approaches deliver similar results as observed in the previous variant. After second 10, both approaches drop in bandwidth. Native QUIC again drops to 180 Mbit/s, SCION QUIC drops to 150 Mbit/s on both paths. During the congestion event, SCION QUIC is slightly below the anticipated factor of 2 in terms of bandwidth utilization compared to Native QUIC. After the congestion event finishes, both approaches return to the same performance as previously observed.

5.3.3 Summary

These experiments attempt a fair comparison between path-aware SCION QUIC and the path-unaware Native QUIC. Fairness is ensured by reducing the available bandwidth to make both approaches depend on the network conditions. To add further realism, congestion events were introduced during the download. In the first variant, traffic was sent over one path. Later, traffic was sent over both paths.

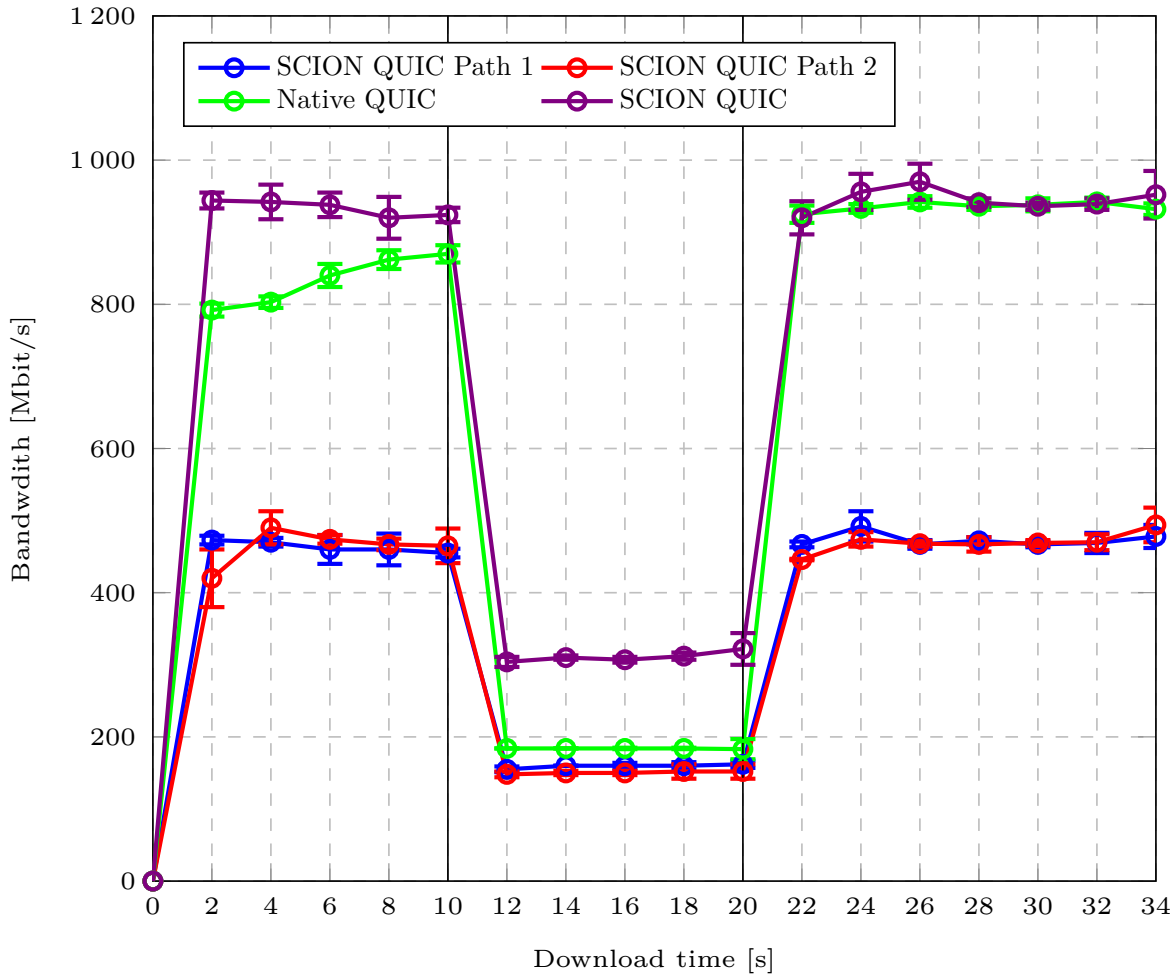

Figure 5.7: Comparison of the download bandwidth of SCION Multipath JF and Native QUIC downloading a 5 Gigabyte torrent file with 5 repetitions expanded by the resulting standard deviation. Between second 10 and 20 both paths are congested with traffic, indicated by the vertical black lines.

In case of one congested path, SCION QUIC demonstrates that it is capable of shifting more of the download bandwidth to the free path. Consequently, it clearly outperforms the Native QUIC approach. SCION QUIC reaches nearly constant bandwidth of around 900 Mbit/s, whereas Native QUIC drops down to 180Mbit/s in the congestion period. Also for 2 congested paths, SCION QUIC nearly reaches the expected factor of 2 in utilized bandwidth of Native QUIC in the congestion period. These experiments thereby demonstrate, that the multipath benefits of SCION outperform a comparable singlepath approach.

5.4 SCION Performance Evaluation

The final section of the evaluation targets to evaluate the maximum performance that Bittorrent over SCION can achieve. As best performing SCION candidate, SCION Multipath JF is used to perform the following measurements. The experiments shown before all used go-torrents the default request strategy. Furthermore, only a single file was downloaded each time. Therefore, in the following, the performance of the other request strategies will be evaluated along with a varying number of files and different file sizes using SCION Multipath JF. The following experiments are performed with the benchmark setup already presented in chapter 4, using both available paths. No artificial bandwidth limitation is imposed on any of the paths.

To eliminate possible bottlenecks in the Bittorrent protocol stack, a performance measurement of SCION with a dedicated bandwidth testing tool was performed. This application was written from scratch to use raw UDP connections over SCION without the overhead of SQUIC. Using this test application, a maximum bandwidth of around 2 Gbit/s over SCION was measured. Detailed measurement are shown in Table 5.4.

Connections	Average Bandwidth [Gbit/s]	Standard deviation[Gbit/s]
2	1.99	0.01
4	2.01	0.01
6	2.00	0.01
8	2.02	0.02

Table 5.4: Average download bandwidth and its standard deviation in Gbit/s of a SCION bandwidth test application.

The measured bandwidth also did not seem to be limited by CPU processing power. The bottleneck seems to originate in the specific implementation of the SCION components, since packets have to traverse the dispatcher and the border router, which causes some delay. One must consider some additional overhead of QUICs congestion control and session management. In contrast to a pure bandwidth test, Bittorrent also has additional protocol overhead. Therefor, less than 2 Gbit/s are expected when the full protocol stack is used.

5.4.1 Request Strategies

At first, go-torrents different request strategies are evaluated. This is done by downloading a 5 Gigabyte file in 5 repetitions. The different request strategies are expected to deliver similar performance, because no network limitations are present and only one seeder provides all pieces of the file. Figure 5.8 shows the download bandwidth of the 3 request

strategies over time. Since their download time is nearly identical with 32 seconds, the actual time is shown on the x-axis instead of the download process in percent.

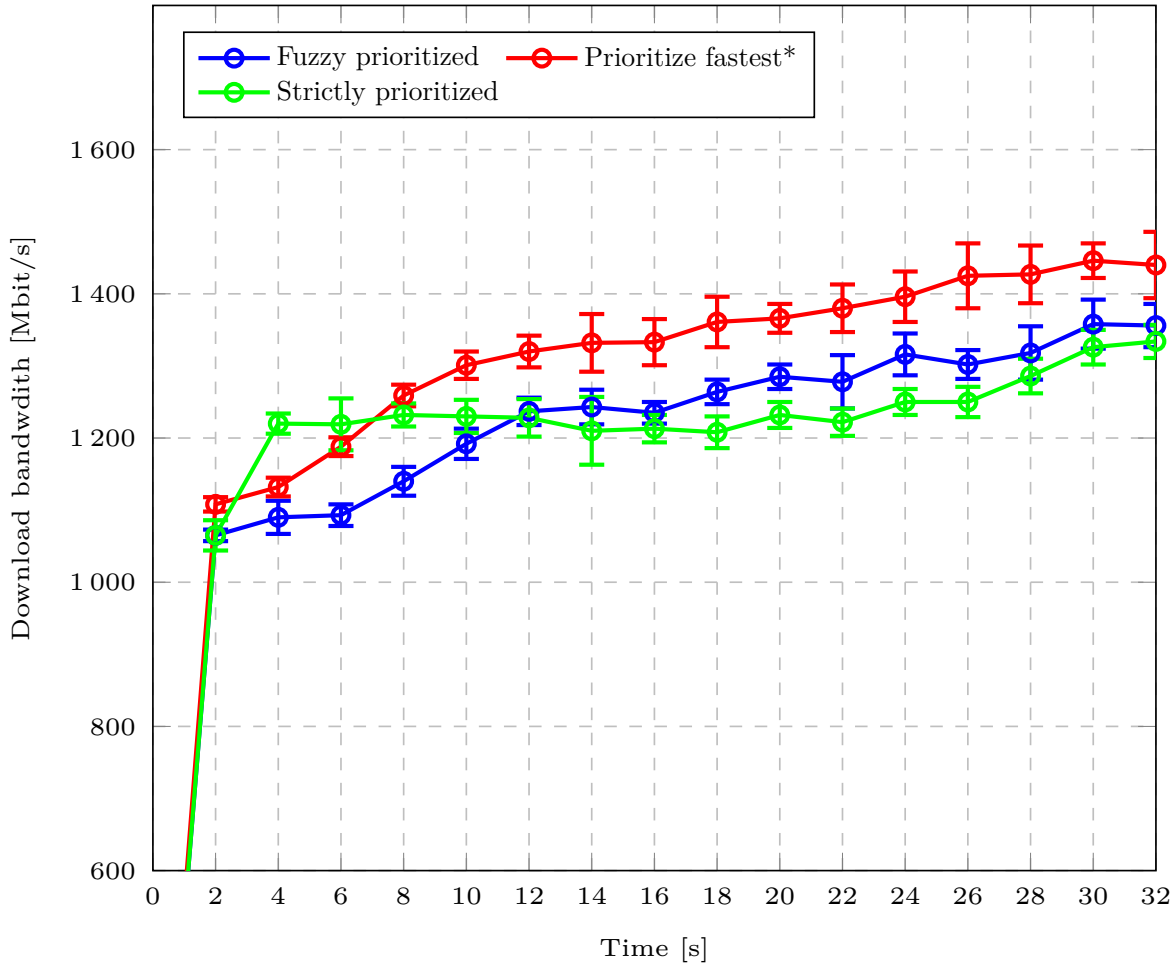

Figure 5.8: Comparison of the presented request strategies for downloading 5 Gigabyte torrent file with 5 repetitions expanded by the resulting standard deviation. *Default request strategy, used for all previous measurements.

All request strategies quickly achieve a bandwidth of around 1100 Mbit/s which further increases over time before the the download is finished. The *Strictly Prioritized* strategy performs a slightly faster start, whereas the increase until the end of the download is lower. *Fuzzy Prioritized* and *Prioritize Fastest* show a similar development over time, whereas *Prioritize Fastest* reaches around 100 Mbit/s more bandwidth. As expected, the request strategies deliver similar results. In summary, no significant improvement is observed by changing the request strategy. Consequently, *Prioritize Fastest* is still used for following experiments.

5.4.2 Varying number of Concurrent Downloads

While downloading a single file, piece processing may be a limiting factor for the rate of new piece requests. Therefore, the bandwidth performance of multiple parallel file downloads is measured. To exclude potential limitations of a single process, each file is downloaded via a dedicated process. With increasing number of concurrent downloads, higher bandwidths are expected, until a threshold is reached, where more torrents not lead to higher bandwidth. The maximum download performance is expected to reach up to 2 Gbit/s minus the overhead of QUIC. As an initial metric for the performance, the download time is measured. Table 5.5 shows the average time required until all files are completely downloaded.

Number of files	Average download time [s]	Standard deviation[s]
1	32.3	0.45
2	51.5	0.5
3	74.2	0.7
5	119.5	1.5
10	245.8	6.5

Table 5.5: Download time and its standard deviation in seconds of a varying number of files downloaded in parallel.

Compared to the download of a single file, the average download time grows by significantly less than a factor of 2 when a second concurrent download is performed. This effect persists for downloading up to 5 files. The download of 10 torrents in parallel does not show further performance increase. For all variants, the standard deviation is very low. This indicates that no file is clearly privileged to the others. Furthermore, the results show that for 10 files the performance remains reasonable stable across the repeated experiment runs.

To further analyze the performance, the actual aggregated download bandwidth is measured. For increasing numbers of up to 5 files, a bandwidth improvement up to a maximum of 2 Gbit/s is expected. However, downloading 10 files in parallel is expected to result in any further improvements. Figure 5.9 shows the aggregated bandwidth of all files over the download completion in percent, in order to achieve comparability despite significantly different download times.

As expected, downloading 2 files in parallel reaches significantly higher bandwidth of around 1700 Mbit/s compared to a single file download with only up to 1400 MBit/s. Increasing the number of concurrent downloads up to 5 torrents results bandwidths of around 1800 Mbit/s, which is close to the expected maximum of 2 Gbit/s minus the QUIC overhead. Downloading 10 torrents in parallel reaches significantly lower bandwidth compared to 5 files. This is assumed to be the result of the CPU overhead caused by

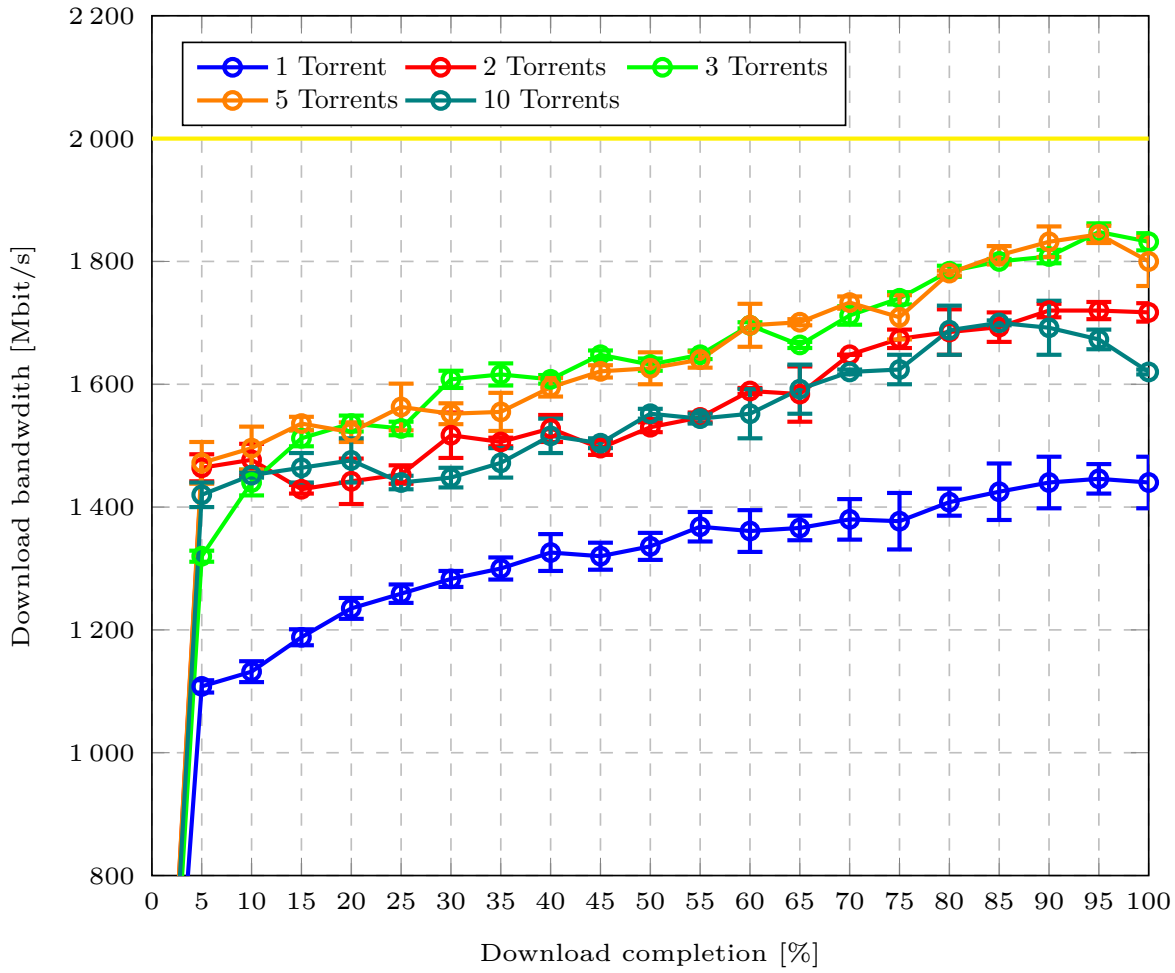

Figure 5.9: Comparison of Bittorrent over SCION downloading up to 10 files (5 Gigabyte) with 5 repetitions expanded by the resulting standard deviation with respect to the measured 2 Gbit/s SCION bandwidth limitation.

Bittorrent processing more files. Furthermore, the bandwidths for 5 and 10 parallel downloads decrease slightly in the last 10% of the download. This is assumed to be due to small differences in completion time of the particular files, which shows more noticeable for higher numbers of files.

To confirm the assumption that additional CPU overhead leads to limited bandwidth when downloading 10 files in parallel, the CPU usages of the variants are measured and analyzed. Figure 5.10 shows the overall CPU utilization of downloading a varying number of files in parallel.

According to the these measurements, downloading more than one file in parallel leads to significantly higher CPU usage. Although the download of 3 files reaches higher bandwidth compared to 2 files, their CPU usage is nearly identical between 60% and 70%. Furthermore, downloading 5 files consume 5% to 10% more CPU resources than 3 files while resulting in similar download bandwidth. Finally, downloading 10 files results in

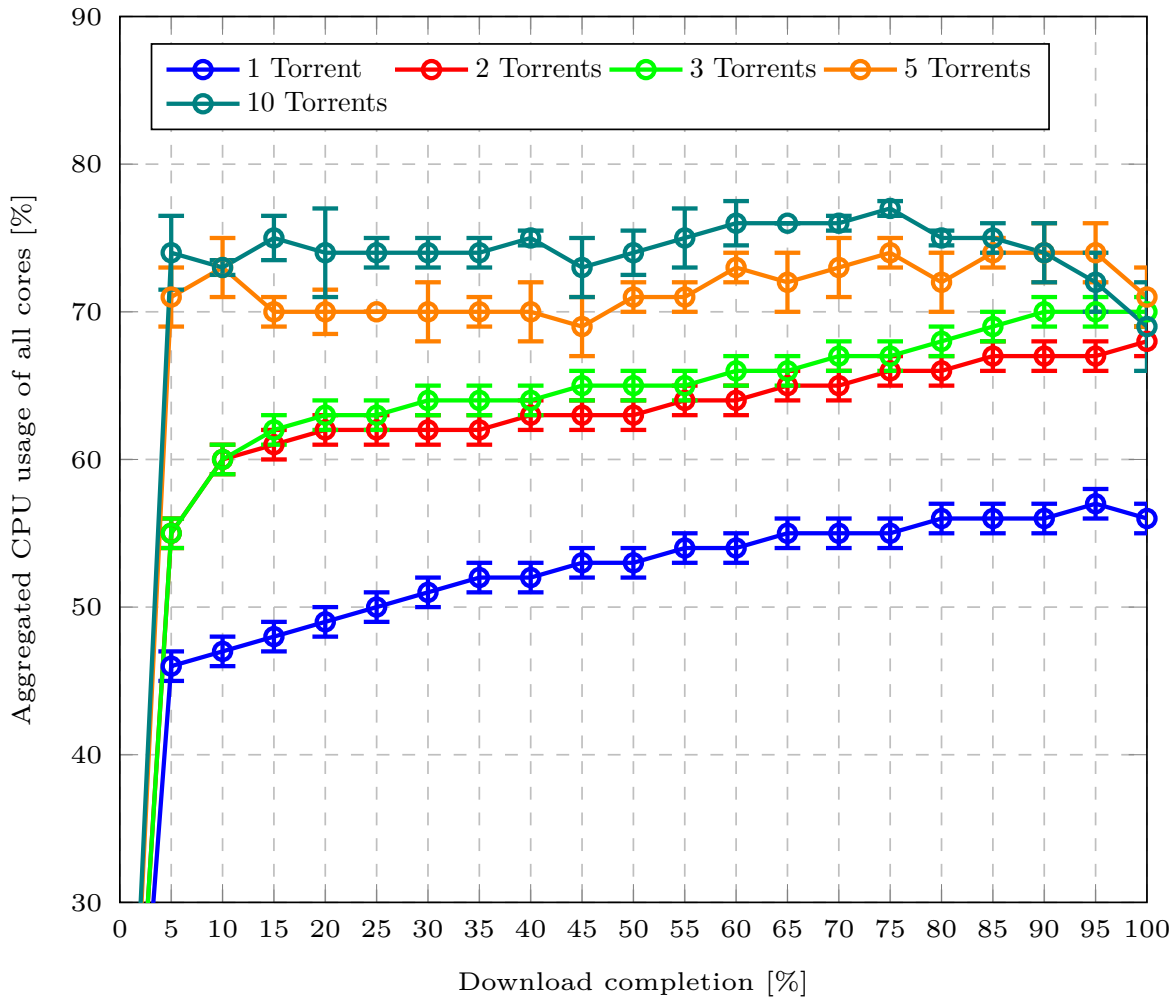

Figure 5.10: Comparison of the overall CPU usage of Bittorrent downloading up to 10 files (5 Gigabyte) with 5 repetitions expanded by the resulting standard deviation.

the highest CPU usage of 75%, while performing similar to 2 files in terms of bandwidth. This confirms the assumption that raising the number of files results in a bandwidth threshold.

5.4.3 Varying File size

As observed from the previous measurements, the bandwidth of SCION Multipath JF increases over time until the download is finished. This was observed for downloading 5 Gigabyte files. One could suspect that larger files experience an earlier bandwidth increase compared to the overall download completion, meaning the maximum bandwidth is reached earlier in the download process. Consequently, a varying file size is evaluated next. According to the measured SCION limitations, a maximum bandwidth of 2 Gbit/s minus the QUIC overhead is expected again. Furthermore, a faster bandwidth increase in

relation to the overall download duration is expected for larger files. Table 5.6 shows the time required to download 3 torrents of varying file size in parallel.

File size [Gigabyte]	Average download time [s]	Standard deviation[s]
1	13.2	0.2
2.5	32.3	0.45
5	74.2	0.7
10	167	3.4
20	409	6.9

Table 5.6: Download time and its standard deviation in seconds of a 3 files downloaded in parallel with varying file sizes.

Starting from 32.3 seconds for a 2.5 Gigabyte file, an increase of the download time by a factor of 2-3 is perceptible with every doubling the file size until 409 seconds are reached when downloading 3 20 Gigabyte files. For larger files, an increased standard deviation can be observed. However, compared to the altitude of the download times, the deviations remain marginal throughout.

For a more detailed analysis, the actual bandwidths are measured over time. Larger files are expected to experience an earlier increase in download bandwidth. Figure 5.11 shows the bandwidth over the download completion, to provide a unified visualization despite the highly different times required for downloading 3 files of different sizes.

In contrast to the expectation of an earlier bandwidth increase for larger files, all variants show similar development, whereas the smallest file size performs the fastest start. For 5 Gigabyte and larger files, a similar bandwidth increase over time is observed. Consequently, bandwidth increase depends on the download progress, not on the actual elapsed time. A possible explanation for this behavior lies in the Bittorrent implementation, which performs countless iterations over an array containing the open pieces. For smaller files, these iterations are shorter, which may reason the faster start for smaller files. However, not a direct leverage is observed, since 5, 10, and 20 Gigabyte files all show a similar bandwidth increase. Thus, further research is needed to explain this behavior.

5.4.4 Summary

The first step in this experiment analyzes the performance of the provided request strategies. It was observed that changing the default request strategy, *Prioritize Fastest*, does not deliver a considerable performance increase. Next, the performance of Bittorrent over SCION was evaluated with a varying number of parallel downloads. With increasing number of files, higher bandwidths were observed, whereas the maximum is reached with

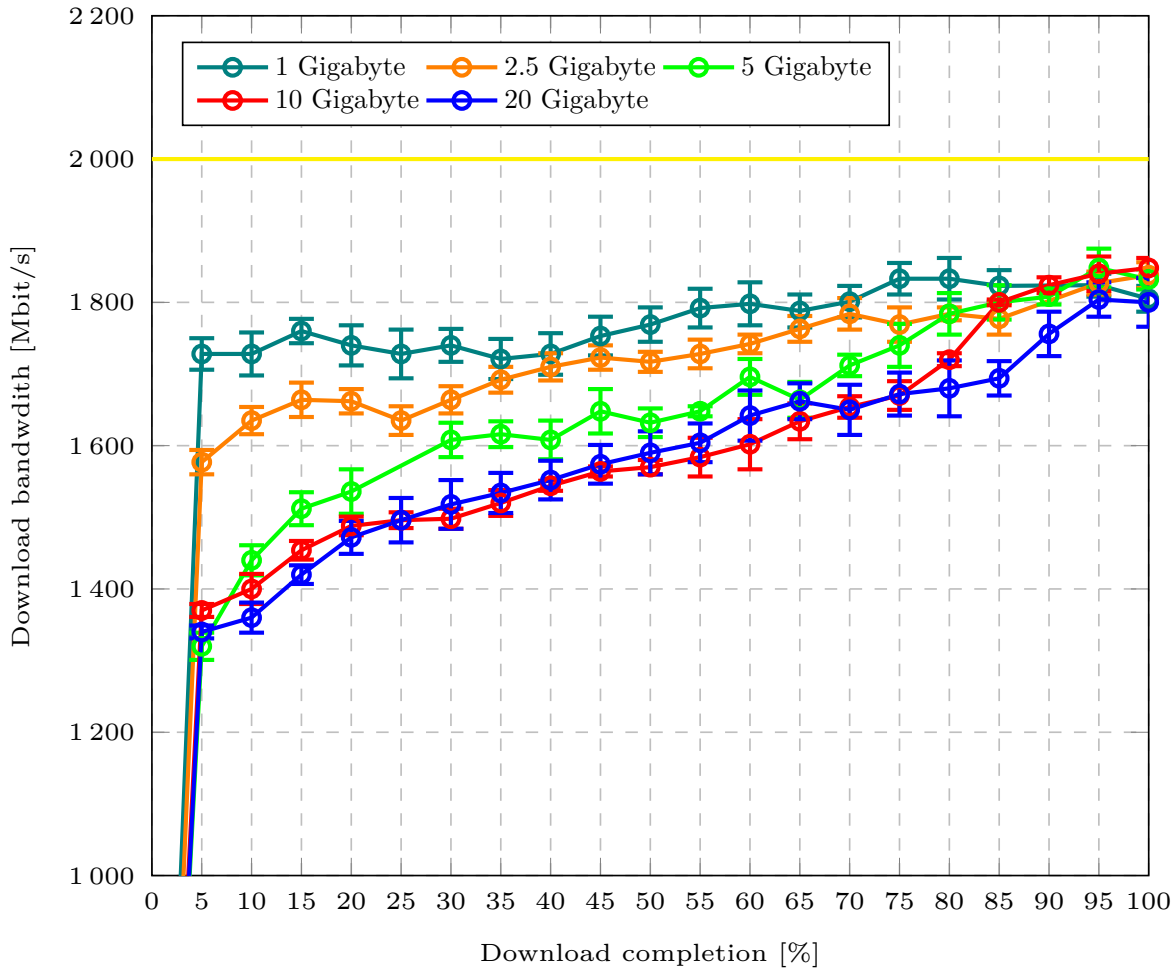

Figure 5.11: Comparison of downloading 1, 2.5, 5, 10, and 20 Gigabyte files (each time 3 downloads in parallel) with 5 repetitions expanded by the resulting standard deviation with respect to the measured 2 Gbit/s SCION bandwidth limitation.

downloading between 3 and 5 files in parallel. Downloading more files lead to lower bandwidth along with increased CPU usage, indicating more CPU overhead on Bittorrent side. Finally, different file sizes were evaluated in order to examine an earlier bandwidth increase in the download process of larger files. The measured results show a similar bandwidth development across all file sizes. Consequently, the bandwidth increase does not appear to depend on the elapsed time, but on the degree of completion of the download process.

5.5 Summary

This chapter covers three experiments consisting of several steps to evaluate Bittorrent over SCION. In the first experiment, the performance and CPU consumption of all presented SCION candidates is evaluated. It was observed, that the HSR delivers not the expected

performance increase. Consequently, not only the open source border router limits the overall bandwidth. An additional bottleneck is potentially located in the rest of the SCION stack, especially assumed in the SCION dispatcher. Furthermore, enabling jumbo frames does not lead to the expected 6 fold performance increase. Instead, a bandwidth increase of only a factor of 2-3 can be observed. Using jumbo frames consumes less CPU usage than the comparable traditional and HSR candidates. As can be observed when using jumbo frames, the CPU utilization of Bittorrent over SCION depends on the number of processed packets, not on the packet size. Finally, the performance difference between the multipath approaches and their singlepath counterparts were lower than expected, due to the assumed bottleneck in the dispatcher.

The second experiment covers comparing of the best SCION candidate against Native QUIC, a path-unaware approach based on UDP. Both approaches use QUIC as transport protocol to provide a fair comparison. Furthermore, the available bandwidth of the benchmark setup was limited to create network bound conditions for a fair setup of both approaches. Traffic was sent over the path to create realistic conditions. The experiment consists of two variants: At first one path is congested with traffic. Then, both paths are congested. By congesting one path, SCION achieves nearly constant download performance shifting the bandwidth to the free path, while Native QUIC drops bandwidth. When congesting both paths, SCION outperforms Native QUIC by aggregating the remaining bandwidth on both paths. In both cases, SCION outperforms Native QUIC.

The final experiment was designed to determine the maximum bandwidth of Bittorrent over SCION. At first, the provided request strategies were evaluated. It turns out that changing the default request strategy does lead to a significant performance increase. Afterwards, Bittorrent over SCION was evaluated with a varying number of files and different file sizes. A maximum bandwidth of 1800 MBit/s was measured by downloading between 3 and 5 files in parallel. Compared to the maximum achievable bandwidth of SCION (2 Gbit/s) and considering the overhead of QUIC, Bittorrent over SCION comes close to this limit. Downloading more than 5 files lead to higher CPU usage without increasing the actual download bandwidth. Changing the file size does not lead to further performance improvements. Furthermore, the bandwidth increase over the download process is not reached faster in the download process for larger files.

Recapitulating, adding multipath support in combination with jumbo frames to Bittorrent over SCION leads to significant performance improvements compared to the traditional singlepath implementation. The assumed bottleneck in the open source border router was verified by replacing it with the HSR. However, the measured performance improvements are significantly lower than expected, especially the performance of SCION Multipath compared to SCION Singlepath. This is reasoned by an additional bottleneck in the SCION stack, assumed probably in the SCION dispatcher. Also the full expected improvements of

using jumbo frames could not be demonstrated. Finally, a reasonable performance increase by aggregating multiple paths was only observed using jumbo frames while being limited by this additional bottleneck.

6 Related Work

Today, file sharing accounts for huge amounts of data transferred over the Internet [34]. As file transfer protocol, Bittorrent is still running over the Internet and has a popular user community [35]. Bharambe et al. discuss Bittorrents network performance mechanisms [7]. They analyze e.g. if download rates of clients can be optimized and if the achieved fairness in a Bittorrent system is sufficient, meaning clients reach similar download and upload rates. Bharambe et al. present that Bittorrents becomes unfair in heterogeneous setups of high bandwidth peers connecting to low bandwidth peers. They show and discuss own approaches that provide a higher level of fairness in those setups. Xia et al. provide a survey of Bittorrents performance [36], analyzing existing Bittorrent features e.g. piece selection. In addition, they present own approaches to address free riding and to improve fairness. Thommes et al. evaluate the promised fairness of Bittorrent [37]. They present three different features to improve Bittorrents fairness and their impact. Furthermore, the features are weighed according to their degree of difficulty to implement them.

File transfer to a huge number of peers offers research possibilities. A lot of research was done with focus on IP multicast as an efficient way to distribute content to multiple peers without duplicating the traffic [38]. Since IP multicast requires a lot of state in switches and routers, the global distribution suffers [39, 40]. Furthermore, IP multicast is rather be used in static setups like video streaming [41] than for highly dynamic on-demand setups like Bittorrent.

As alternative to IP multicast, overlay approaches were presented. Bullet [42] is an overlay approach to efficiently distribute files from a single source to a huge number of receivers introduced by Kostić et al. Similar to Bittorrent, files are split into pieces and distributed to other peers. Bullet uses an overlay tree to initially create the overlay network and then moves to an overlay mesh. For file distribution, disjoint pieces of a new file are sent to all overlay participants. All participants request pieces of the file from each other until each of them has all pieces. Kostić et al. present that using Bullet on a random tree can increase throughput by factor two compared to using a traditional bandwidth tree. Another file distribution approach introduced by Cherkasova et al. is FastReplica [43], which aims to provide reliable and efficient file replication into the Internet. Similar to Bittorrent, files are split into pieces and the pieces are transferred to nodes within a replication group. These nodes use all available paths in the replication group to download all missing pieces

from the other participants. Both approaches share similarities with Bittorrent, especially in piece-based file distribution. Nevertheless, both implement a single source principle, in contrast to Bittorrents where all peers can add files to the setup. Consequently, problems like fairness and free riding are omitted in these approaches.

With the tremendously growing amount of data, also the requirement of security in the Internet architecture grows. Focusing on transparent path control from the sender, similar approaches to SCION were presented. Godfrey et al. introduce PathLet Routing [44], an approach based on segmenting a route into path fragments. Similar to SCION, PathLet tackles problems of inter-domain routing. In both approaches, path fragments contain a next hop and a list of remaining interface ids. Also the forwarding approach of both is quite similar, they use the next hop field to determine where to forward the packet. In the case of SCION, the next hop is represented with egress and ingress interfaces. PathLet stores a forward rule in the next hop entry. Also NIRA [3] introduced by Yang proposes routing control by the sender. Similar to SCION, NIRA is concentrated on determining and applying a list of ASes that a packet traverses to its destination. As extension for SCION, Basecu et al. presented SIBRA [5] an interesting approach to protect modern Internet architectures against DDoS attacks. SIBRA follows the approach of bandwidth reservation for inter-domain paths to ensure functionality of those paths.

The initial implementation of Bittorrent over SCION was presented by Koppehel [9], consisting of a working implementation and various benchmarks. Another approach of transferring and storing data over SCION is SCION Swarm [45], introduced by Ryll. SCION Swarm also implements a path-aware solution of file distribution. The achieved performance was less than on an IP-based setup due to the additional computing effort required by the SCION stack. Furthermore, SCION was in an early stage at the time SCION Swarm was evaluated. Frei et al. present Hercules [46], a setup for high performance bulk data transfer over SCION reaching around 30Gbit/s on a 40Gbit/s link excluding disk I/O. Hercules bypasses the SCION dispatcher to achieve this magnitude of performance. Consequently, bypassing the dispatcher promises also significant improvements for Multipath Bittorrent over SCION.

Establishing multipath data transfer can also be realized completely on application level. Yu et al. presented mPath [1], an algorithm and implementation using proxies to create multiple paths to particular end hosts. The underlying assumption of mPath is the existence of less congested paths then the direct one. Mpath uses similar approaches as SCION consisting of e.g. path lookup. Yu et al. compare mPath to TCP in various setups, detecting that mPath is slower having a small number of node pairs. Furthermore, they inspect that increasing distance between sender and receiver leads to higher performance improvements for mPath.

7 Conclusion and Future Work

7.1 Conclusion

The aim of this work is to evaluate the research question to what extent the SCION multipath architecture can improve the existing Bittorrent over SCION implementation. To answer this question, a multipath-aware Bittorrent over SCION was designed, implemented and evaluated. The evaluation was separated into analyzing the performance of different SCION candidates and torrent configurations. Furthermore, a comparison against a path-unaware approach was performed.

Considering the overall download performance, multipath aggregation was successfully implemented and evaluated. SCION Multipath JF performs best and reaches performance improvement of 300% compared to the existing singlepath implementation. However, this improvement is mainly reached by enabling jumbo frames. The actual bandwidth increase achieved by adding multipath support results in 25% for the traditional candidates, 40% with jumbo frame usage and 50% by running the HSR. This is far under the expected values due to an assumed additional bottleneck in the SCION dispatcher. Nevertheless, Multipath Bittorrent over SCION is able to handle congested paths and concentrate the download bandwidth to the remaining paths. In a fair comparison, Multipath Bittorrent over SCION clearly outperforms a path-unaware approach.

Recapitulating, the assumed bottleneck in the SCION dispatcher prevents Multipath Bittorrent over SCION to reach the expected performance increases compared to singlepath approaches. While being limited by this bottleneck, the SCION multipath architecture improves the existing Bittorrent over SCION implementation only to a moderate extent, which increases by enabling jumbo frames. However, overcoming this bottleneck promises to reach the expected performance increase by making Bittorrent over SCION benefit entirely from the SCION multipath architecture.

7.2 Future Work

To improve the performance of Multipath Bittorrent over SCION, the assumed bottleneck in the dispatcher is planned to be analyzed and bypassed. After overcoming this bottleneck,

tests with larger setups and more heterogeneous paths are planned. Furthermore, a heuristic approach to add previously discarded paths to the path set is planned. Instead of closing discarded connections, a waiting state will be introduced to avoid repeated handshakes for the same path. Furthermore, the path selection algorithm can be adapted to Bittorrents piece request strategies. In endgame mode, using more paths could result in faster finish of the download. However, in the other strategies using less paths to concentrate the bandwidth could be more beneficial.

In case multiple paths are discarded by the path selection algorithm, these can be inspected to gather information about possible shared bottlenecks. A possible approach to identify them is to look for repeatedly occurring ingress or egress interfaces. If an interface occurs in a sufficiently high number of paths, a shared bottleneck is assumed and not yet tested paths that contain this interface could be skipped.

Another improvement for Bittorrent over SCION is the missing tracker implementation. SCION already provides libraries and examples for setting up HTTP clients and servers, called SCION HTTP [47]. Consequently, a tracker implementation for SCION peers based on SCION HTTP is planned. Regarding the tracker design, its worth evaluating a possible preselection of paths by the tracker that returns a sorted list of path-level peers. In case the tracker is filled with the gathered data from the peers, this approach may benefit of shared knowledge about paths.

Finally, an existing multipath socket [48] can be integrated into Bittorrent over SCION. This approach enables path failover on a dedicated multipath connection. Consequently, this comparison can provide useful information to improve the multipath support of Bittorrent over SCION.

Bibliography

[1] Xu, Yin ; Leong, Ben ; Seah, Daryl ; Razeen, Ali: mPath: High-bandwidth data transfers with massively multipath source routing. In: *IEEE Transactions on Parallel and Distributed Systems* 24 (2012), Nr. 10, S. 2046–2059

[2] Gvozdiev, Nikola ; Vissicchio, Stefano ; Karp, Brad ; Handley, Mark: On low-latency-capable topologies, and their impact on the design of intra-domain routing. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, S. 88–102

[3] Yang, Xiaowei: NIRA: A new Internet routing architecture. In: *ACM SIGCOMM Computer Communication Review* 33 (2003), Nr. 4, S. 301–312

[4] Rothenberger, Benjamin ; Roos, Dominik ; Legner, Markus ; Perrig, Adrian: PISKES: Pragmatic Internet-Scale Key-Establishment System. In: *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS'20)*, 2020

[5] Basescu, Cristina ; Reischuk, Raphael M. ; Szalachowski, Pawel ; Perrig, Adrian ; Zhang, Yao ; Hsiao, Hsu-Chun ; Kubota, Ayumu ; Urakawa, Jumpei: SIBRA: Scalable internet bandwidth reservation architecture. In: *arXiv preprint arXiv:1510.02696* (2015)

[6] Zhang, Xin ; Hsiao, Hsu-Chun ; Hasker, Geoffrey ; Chan, Haowen ; Perrig, Adrian ; Andersen, David G.: SCION: Scalability, control, and isolation on next-generation networks. In: *2011 IEEE Symposium on Security and Privacy* IEEE, 2011, S. 212–227

[7] Bharambe, Ashwin R. ; Herley, Cormac ; Padmanabhan, Venkata N.: Analyzing and improving a bittorrent networks performance mechanisms. In: *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications* IEEE, 2006, S. 1–12

[8] Bharambe, Ashwin ; Herley, Cormac ; Padmanabhan, Venkata N.: Understanding and deconstructing bittorrent performance. In: *Proc. ACM Sigmetrics*, 2005

[9] Koppehel, Martin: *Case Study of BitTorrent over SCION*. https://mko.dev/research/scion-bittorrent.pdf. – 2020-11-09

[10] Cohen, Bram: *The BitTorrent Protocol Specification*. https://www.bittorrent.org/beps/bep_0003.html. – Accessed 2020-10-30

[11] Cohen, Bram: Incentives build robustness in BitTorrent. In: *Workshop on Economics of Peer-to-Peer systems* Bd. 6, 2003, S. 68–72

[12] Network Security Group, ETH Zurich: *SCION Internet Architecture*. https://github.com/scionproto/scion. – Accessed 2020-08-24

[13] Kaspersky Labs: *IT Security Risks Survey 2017*. https://www.msspalert.com/cybersecurity-research/kaspersky-lab-study-average-cost-of-enterprise-ddos-attack-totals-2m/. – Accessed 2020-08-23

[14] Langley, Adam ; Riddoch, Alistair ; Wilk, Alyssa ; Vicente, Antonio ; Krasic, Charles ; Zhang, Dan ; Yang, Fan ; Kouranov, Fedor ; Swett, Ian ; Iyengar, Janardhan u. a.: The quic transport protocol: Design and internet-scale deployment. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, S. 183–196

[15] Network Security Group, ETH Zurich: *RAINS, Another Internet Naming Service*. https://github.com/netsec-ethz/rains. – Accessed 2020-10-16

[16] Kreutz, Diego ; Ramos, Fernando M. ; Verissimo, Paulo E. ; Rothenberg, Christian E. ; Azodolmolky, Siamak ; Uhlig, Steve: Software-defined networking: A comprehensive survey. In: *Proceedings of the IEEE* 103 (2014), Nr. 1, S. 14–76

[17] Perrig, Adrian et a.: *SCION: A Secure Internet Architecture*. https://www.scion-architecture.net/pdf/SCION-book.pdf

[18] Akdemir, Kahraman ; Dixon, Martin ; Feghali, Wajdi ; Fay, Patrick ; Gopal, Vinodh ; Guilford, Jim ; Ozturk, Erdinc ; Wolrich, Gil ; Zohar, Ronen: Breakthrough AES performance with intel AES new instructions. In: *White paper, June* (2010), S. 11

[19] Zurich, ETH: *SCIONLab*. https://www.scionlab.org/. – Accessed 2020-08-24

[20] anacrolix: *Full-featured BitTorrent client package and utilities*. https://github.com/anacrolix/torrent. – Accessed 2020-08-22

[21] Norberg, Arvid: *uTorrent transport protocol*. https://www.bittorrent.org/beps/bep_0029.html. – Accessed 2020-10-15

[22] Stevens, Wright u. a.: TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. (1997)

[23] Johnson, Ben: *Boltdb - An embedded key/value database for Go*. https://github.com/boltdb/bolt. – Accessed 2020-10-15

[24] Shaw, Evan: *A portable mmap package for Go*. https://github.com/edsrzf/mmap-go. – Accessed 2020-10-16

[25] anacrolix, martin31821: *Full-featured BitTorrent client package and utilities*. https://github.com/martin31821/torrent/. – Accessed 2020-08-14

[26] ethz netsec: *SCION appquic package*. https://github.com/netsec-ethz/scion-apps/tree/master/pkg/appnet/appquic

[27] lucasclemente: *A QUIC implementation in pure go*. https://github.com/lucas-clemente/quic-go. – Accessed 2020-08-18

[28] Network Security Group, ETH Zurich: *SCION Internet Architecture - Posix Router*. https://github.com/scionproto/scion/tree/master/go/posix-router. – Accessed 2020-10-17

[29] Systems, Anapaya: *Anapaya Systems*. https://www.anapaya.net/. – Accessed 2020-08-22

[30] Linux Foundation: *Dataplane development kit*. https://www.dpdk.org/. – Accessed 2020-10-12

[31] Linux Foundation: *VPP DPDK tutorial*. https://wiki.fd.io/view/VPP/Tutorial_DPDK_and_MacSwap. – Accessed 2020-10-12

[32] Linux Foundation: *VPP/What is VPP*. https://wiki.fd.io/view/VPP/What_is_VPP%3F. – Accessed 2020-10-12

[33] Linux Foundation: *Flow Bifurcation How-to Guide*. https://doc.dpdk.org/guides-17.05/howto/flow_bifurcation.html. – Accessed 2020-10-12

[34] statista: *File sharing data traffic volume in western europe*. https://www.statista.com/statistics/267205/file-sharing-data-traffic-volume-in-western-europe/. – Accessed 2020-10-14

[35] Smith, Graig: *BitTorrent Statistics and Facts (2020)*. https://expandedramblings.com/index.php/bittorrent-statistics-facts/. – Accessed 2020-10-14

[36] Xia, Raymond L. ; Muppala, Jogesh K.: A survey of bittorrent performance. In: *IEEE Communications surveys & tutorials* 12 (2010), Nr. 2, S. 140–158

[37] Thommes, Richard ; Coates, Mark: Bittorrent fairness: analysis and improvements. In: *Proc. Workshop Internet, Telecom. and Signal Proc*, 2005

[38] Deering, Steve E.: *RFC1112: Host Extensions for IP multicasting.* 1989

[39] Diot, Christophe ; Levine, Brian N. ; Lyles, Bryan ; Kassem, Hassan ; Balensiefen, Doug: Deployment issues for the IP multicast service and architecture. In: *IEEE network* 14 (2000), Nr. 1, S. 78–88

[40] Ratnasamy, Sylvia ; Ermolinskiy, Andrey ; Shenker, Scott: Revisiting IP multicast. In: *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, 2006, S. 15–26

[41] Harte, Lawrence: *Introduction to Data Multicasting, IP Multicast Streaming for Audio and Video Media Distribution.* Althos, 2008

[42] Kostić, Dejan ; Rodriguez, Adolfo ; Albrecht, Jeannie ; Vahdat, Amin: Bullet: High bandwidth data dissemination using an overlay mesh. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, S. 282–297

[43] Cherkasova, Ludmila ; Lee, Jangwon: FastReplica: Efficient Large File Distribution Within Content Delivery Networks. In: *USENIX Symposium on Internet Technologies and Systems*, 2003, S. 85–98

[44] Godfrey, P B. ; Ganichev, Igor ; Shenker, Scott ; Stoica, Ion: Pathlet routing. In: *ACM SIGCOMM Computer Communication Review* 39 (2009), Nr. 4, S. 111–122

[45] Ryll, Leopold: *Development and Evaluation of a SCION-Enhanced Distributed Storage Solution.* 2018

[46] Mattias Frei, Francois W.: *Hercules - Bulk Data Transfer over SCION.* `https://www.scion-architecture.net/pages/scion_day/slides/Hercules%20-%20Bulk%20Data%20Transfer%20over%20SCION.pdf`. – Accessed 2020-11-08

[47] Network Security Group, ETH Zurich: *HTTP over SCION/QUIC.* `https://github.com/netsec-ethz/scion-apps/tree/master/pkg/shttp`. – Accessed 2020-10-17

[48] Network Security Group, ETH Zurich: *Mpsquic.* `https://github.com/FR4NK-W/scion-apps/tree/mpsquic_lean_unbundle/lib/mpsquic`. – Accessed 2020-10-17

A Appendix

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ausschließlich zu Zwecken der Plagiatsüberprüfung und zur Optimierung der Leistungsbewertung und –kontrolle meiner Arbeit durch die Otto-von-Guericke-Universität Magdeburg (OvGU) bzw. von dieser beauftragte Dritte (z. B. *iParadigms Europe Limited*, Anbieter der Plagiatsprüfungssoftware "Turnitin") räume ich der OvGU an meiner Arbeit ein zeitlich und örtlich unbeschränktes, einfaches, unentgeltliches Nutzungsrecht ein, das es gestattet, die Arbeit zu den vorgenannten Zwecken zu vervielfältigen, zu speichern bzw. zu archivieren.

Marten Gartner,
Magdeburg, den 11.11.2020