Hummingbird: Fast, Flexible, and Fair Inter-Domain Bandwidth Reservations

Karl Wüst*
Mysten Labs

Giacomo Giuliari*
Mysten Labs

Markus Legner
Mysten Labs

Jean-Pierre Smith
Mysten Labs

Marc Wyss
ETH Zurich

Jules Bachmann
ETH Zurich

Juan A. Garcia-Pardo
ETH Zurich

Adrian Perrig
Mysten Labs

Abstract

To realize the long-standing vision of providing quality-of-service (QoS) guarantees on a public Internet, this paper introduces Hummingbird: a lightweight QoS-system that provides fine-grained inter-domain reservations for end hosts.

Hummingbird enables flexible and composable reservations with end-to-end guarantees, and addresses an often overlooked, but crucial, aspect of bandwidth-reservation systems: incentivization of network providers. Hummingbird represents bandwidth reservations as tradable assets, allowing markets to emerge. These markets then ensure fair and efficient resource allocation and encourage deployment by remunerating providers. This incentivization is facilitated by decoupling reservations from network identities, which enables novel control-plane mechanisms and allows the design of a control plane based on smart contracts.

Hummingbird also provides an efficient reservation data plane, which streamlines the processing on routers and thus simplifies the implementation, deployment, and traffic policing, while maintaining robust security properties. Our prototype implementation demonstrates the efficiency and scalability of Hummingbird's asset-based control plane, and our high-speed software implementation can fill a 160 Gbps link with Hummingbird packets on commodity hardware.

CCS Concepts

• **Security and privacy** → *Security protocols*; *Denial-of-service attacks*; *Distributed systems security*; • **Networks** → *Network protocol design.*

Keywords

Quality-of-Service, Blockchain, QoS, SCION, Sui, Bandwidth Reservations

ACM Reference Format:
Karl Wüst, Giacomo Giuliari, Markus Legner, Jean-Pierre Smith, Marc Wyss, Jules Bachmann, Juan A. Garcia-Pardo, and Adrian Perrig. 2025. Hummingbird: Fast, Flexible, and Fair Inter-Domain Bandwidth Reservations. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3718958.3750495

*Both authors contributed equally to this paper

SIGCOMM '25, Coimbra, Portugal

ACM ISBN 979-8-4007-1524-2/2025/09
https://doi.org/10.1145/3718958.3750495

1 Introduction

Most applications today require stable network connections to provide optimal functionality. This holds true, in particular, for applications that rely heavily on the cloud or that require real-time communication. Video calls, which have become more frequent with the popularity of remote work, are a common example of such applications. Ensuring a high quality of service (QoS) for important calls can have a significant impact on the success of businesses. In online games, the reliability of the connection between players and the game server can influence the outcome of a competition, providing an advantage to players with better network connectivity. The quality of the network connection can also impact traditional finance applications as well as decentralized finance protocols [49]. In both cases, ensuring that a trade is submitted on time can make or break the trade. In B2B communication settings, QoS on a public Internet can satisfy the needs of high-availability use cases and replace the use of SD-WAN (valued at USD 5B in 2023 [42]) or leased line technologies (valued at USD 12B in 2023 [41]). Given the market sizes, incentives exist for challengers to leverage QoS to enter and capture market share.

To fully realize these use-cases, flexible, short-term guarantees for reliable network connectivity across multiple domains in the Internet are required. However, in the current Internet, providing guaranteed QoS is only possible within a centralized domain, i.e., an autonomous system (AS), through expensive leased lines, or SD-WANs. In addition, these guarantees cannot be flexibly obtained for short or medium time periods, i.e., on the order of seconds, minutes, hours, or days.

Previous proposals to provide QoS for Internet traffic [3, 7, 17, 18, 51, 54] have several limitations. Earlier techniques [3, 7, 51] do not consider the presence of adversaries, and are therefore only applicable in intra-domain settings. To withstand powerful network adversaries, more recent proposals [17, 54] enforce reservations through *network capabilities* [2], unforgeable cryptographic tokens that can be checked efficiently and statelessly by routers on the forwarding path. However, the attack resilience of these systems comes at the expense of increased control-plane complexity [17], or inflexible reservation sizes [18, 54] (e.g., the source cannot specify the amount of bandwidth to be reserved). The tradeoff between control-plane complexity and reservation flexibility may appear fundamental: Increasing the degrees of freedom in which reservations can be parameterized and composed *should* come at the cost of

a more intricate control-plane logic; conversely, a more streamlined control plane *should* constrain reservation expressiveness.

This paper moves beyond this impasse by proposing Hummingbird, a system that allows expressive bandwidth reservations on the data plane, while avoiding the coordination and timeliness requirements that encumbered the control plane of previous proposals.

The key insights are that reservations can be made independent from in-band identities, such as network addresses, that they can be granted to the source on a per-hop basis without coordination between on-path ASes, and that they can be negotiated *well ahead of their start time* if desired. The source is then responsible for obtaining and composing reservations to protect end-to-end paths.

This decoupling of the control plane from the data plane enables new control-plane designs that address a fundamental part of a bandwidth-reservation system: A mechanism to incentivize fair and efficient allocation of reservations.

In contrast to previous solutions, we prioritize incentivization by designing a control plane that represents bandwidth as freely tradable assets—that can be sold, bought, and re-sold. This approach enables the creation of bandwidth markets that attach a monetary value to reservations, and ensure that bandwidth supply provided by ASes, and demand originating from hosts, converge to the fair market price. This mechanism serves multiple purposes: (i) incentivize ASes to provide bandwidth reservations to hosts, (ii) maximize the utility of reservations and provide economic fairness, and (iii) ensure that the cost of preventing a host from obtaining a reservation is equal to the market value of the reservation.

We thus design a control plane that makes use of smart contracts to (i) create tradable assets for bandwidth reservations, and (ii) allow obtaining end-to-end QoS guarantees atomically without interaction between on-path ASes.

On the data plane, the sender adds authentication tags to each packet, serving as capabilities that are checked efficiently by border routers on the path. Hummingbird incorporates several optimizations to ensure that reservations granted ahead of time cannot be abused, and that information about the heterogeneous on-path reservations can be transmitted efficiently.

2 Related Work

Several different systems to improve on best-effort traffic were proposed in the 1990s, of which the two most prominent are IntServ [7, 51] and DiffServ [3]. Unfortunately, neither of them is applicable to adversarial settings and both systems suffer from additional shortcomings: IntServ can provide strong guarantees (in the absence of malicious actors) but requires a large amount of per-connection state and has to make complicated decisions on routers when processing RSVP requests [7]. DiffServ, on the other hand, is highly scalable but only provides weak communication guarantees, and it is mainly used to define different traffic types *within* an AS. Thereafter, researchers have studied traffic prioritization protocols that use cryptographic authentication to enforce traffic prioritization [2, 31, 32, 38, 56, 57]. These systems, however, were either not scalable or did not provide sufficient guarantees compared to best effort, and never saw widespread adoption.

Inter-domain bandwidth reservations. The research area has seen a revival in the past decade with the rise of path-aware networking architectures, like SCION [9], which facilitate bandwidth reservations with features such as packet-carried forwarding state and path stability [5, 17, 18, 23, 53, 54]. However, even in this setting, previous proposals make different choices for the trade-off between scalability, deployability, fairness, simplicity, and security. In the following, we describe two state-of-the-art systems and their limitations for our use case.

Colibri [17] provides its inter-domain bandwidth reservations with a two-step process. First, ASes establish segment reservations that cover SCION up-, core-, or down-segments. Second, end-to-end reservations are established by combining such segment reservations, using fractions of the segment reservation bandwidth. Establishing a Colibri reservation is performed by the source AS on behalf of the host using a *gateway*, and requires coordination between all ASes on the path. The gateway manages and monitors reservations for its hosts, which do not have access to the cryptographic keys to authenticate packets themselves. Instead, the gateway embeds the required authenticators in the packets. Due to the complexity of separately monitoring each path, on-path ASes only monitor traffic probabilistically, and, if overuse is detected, punish the source AS, e.g., by declining future reservations.

Further, Colibri requires the deployment of an additional key distribution infrastructure (DRKey [30, 44]), its control plane is complex to implement, and it requires duplicate suppression, i.e. filtering of duplicate packets, on the data plane. We discuss the need for duplicate suppression in related work and why Hummingbird can do without in Section 5.4 in more detail. Additionally, Colibri does not allow any partial reservations, in which bandwidth is only reserved on some of the hops, and thus can only be used if each AS on the path provides a reservation to the host. Finally, Colibri offers little flexibility regarding the validity period of reservations: a reservation is valid as soon as it is received and must be renewed after a fixed time interval of 16 s.

With Helia [54], Wyss et al. introduce *flyover* reservations, which are granted per-AS instead of the full path. This reduces the complexity of reservations compared to Colibri and allows for partial reservations on a path – an approach we adopt in Hummingbird as well. One limitation of Helia is that the reservation granularity is coarse and, most importantly, the reserved bandwidth as well as the start and expiration times cannot be negotiated between the source and the on-path AS. Instead, the reservations have fixed time slots and their size is automatically set to ensure that each source can obtain a reservation. In addition, Helia does not allow end-hosts to create and use reservations directly. Similar to Colibri, it requires an AS-gateway to authenticate packets, it does not allow creating reservations ahead of time, and it also requires the DRKey [30, 44] infrastructure to be in place. Further, Helia does not support atomic end-to-end reservation guarantees, since the reservation for each hop is obtained individually without coordination.

Network utilization and economic incentives. Since the early days of the Internet, researchers explored the use of economic incentives to improve the utilization of network resources [12, 36]. These works were inspired by the recent memory of the "congestion collapses" [24] of some years prior, and asked the question of

whether pricing congestion *explicitly* could help improve the allocation of the scarce bandwidth at the disposal of the early Internet. Kelly et al. [27] demonstrate that TCP-like congestion control algorithms can naturally be interpreted as a price-based rate-allocation mechanism, where the "shadow prices" of are inferred from flow rates. These publications served as the theoretical foundation for the later development of a rich literature on network utility maximization (NUM) through implicit or explicit pricing [16, 37, 40].

NUM-based systems have several key differences from Hummingbird. First, they often consider a network with a single controller, or "one-hop" networks, where the controller can enforce the pricing and billing of resources. Hummingbird, instead, considers the full scope of inter-domain resource billing and enforcement, without a single central authority. Second, given the single-path nature of the Internet, these works only leave two options to the end-host: either to pay the price set by the network, or to reduce their network usage. Hummingbird builds on SCION's multi-path forwarding architecture, allowing end-hosts to reroute their traffic depending on congestion and prices. Further, only reservation-protected traffic—which is a new and opt-in service–is priced through a market in Hummingbird. Therefore, our design does not alter the existing inter-domain economic relationships between ASes, and only adds a new layer of economic incentives for a new service. [1]

Finally, these works are mainly focused on incentive mechanisms, rather than concrete system design; Hummingbird, on the other hand, provides a complete design and implementation of a system that can be deployed in adversarial settings. While our work considers a network and economic model that is very different from those studied in the NUM literature, analyzing Hummingbird under the lens of NUM is a promising direction for future work.

Regarding the use of blockchains as supporting infrastructure for network protocols, Route Bazaar [8] pioneered the use of blockchain to implement decentralized QoS markets. Yet, this work does not provide a concrete system design, and does not consider reservation enforcement nor adversarial action. Hummingbird, on the other hand, provides a complete design and implementation, including reservation monitoring and policing, preventing unauthorized access and overuse.

3 Hummingbird Overview

In this section, we describe the properties and goals of Hummingbird, state the assumptions that we make on the network and infrastructure, and finally provide a high-level overview of the system design.

3.1 Goals & Properties

The goal of Hummingbird is to enable end-to-end QoS guarantees in the Internet through bandwidth reservations. Traffic using reservations is prioritized over best-effort traffic, and is therefore shielded from congestion and DoS attacks.

To that end, we rely on existing intra-domain traffic separation and prioritization mechanisms, e.g., DiffServ, or MPLS tunnels within an AS's network. We focus on coordinating the utilization of these systems in the global inter-domain context, enabling their flexible and scalable composition, and ensuring that QoS-traffic does not suffer from congestion at peering points between ASes.

The core challenges we aim to solve are (i) to scale control and data plane to the size of the Internet; (ii) to avoid reservation spoofing and abuse by adversaries; and, (iii) to provide incentive mechanisms for the fair and efficient allocation of reservations. In particular, we design Hummingbird to provide the following properties:

Independent & Composable Flyover Reservations. Each reservation is granted for an individual AS hop, called a *flyover*, and prioritizes the source's traffic (up to the reserved bandwidth) intra-AS from the ingress to the egress router of the AS, and inter-AS from the egress router to the ingress router of the next AS.These reserved hops can then be composed to obtain end-to-end guarantees, or they can be used independently to only reserve parts of a path that are expected to be congested. This composability also allows reusing a reservation on one hop for connections to different entities. For example, one large reservation at a central AS in the network can be used together with multiple smaller reservations to obtain bandwidth guarantees for connections to multiple destinations.

Control-Plane Independence. The data plane is independent of the control plane; i.e., reservations can be created without relying on the network identity of the source. This decoupling allows each AS to decide how it wants to offer reservations to its customers, and thus—in contrast to existing solutions—enables out-of-band mechanisms such as bandwidth marketplaces.

Atomic Path Reservations. Since reservations are obtained for each hop independently, it may occur that a source can only obtain reservations for part of the path, while on a few key hops there is no bandwidth available. If the source requires full path protection, then the obtained reservations are useless to the source, potentially cause a financial loss, and are no longer available to other hosts, despite being unused. One useful property we require for Hummingbird's control plane is therefore a mechanism to provide *atomic path reservations*, whereby the source is guaranteed to either obtain reservations for all requested hops, or none at all.

Stateless Reservation Authentication. Transit ASes can verify the authenticity of each reservation on the fly, based only on the reservation information in the packet.

Efficient Policing. Monitoring and policing of bandwidth reservations in Hummingbird are feasible with simple methods and *without* relying on duplicate suppression, which is challenging to implement in the network [33]. Our design does not require complex probabilistic monitoring schemes, nor does it rely on punishment to prevent overuse. Instead, traffic can be policed efficiently using deterministic monitoring and requires only storing minimal state. Importantly, each AS can individually decide and limit the number of reservations that it can afford to monitor simultaneously.

Reservation Incentives. An important aspect of a bandwidth reservation is providing incentives for a fair and efficient resource allocation. Since incentivization is usually an afterthought in the design of bandwidth-reservation systems, retrofitting such a mechanism onto the previous proposals creates significant inefficiencies and overhead. Hummingbird is designed to enable efficient

[1] The emergent interactions between the incentive structures of the best-effort Internet and the reservation market are an interesting topic for future work.

incentivization—through markets—with a control plane that represents bandwidth as a tradable asset (see Section 4.2).

3.2 Network & Infrastructure

Our system requires the following minimal assumptions on network stability and infrastructure deployment.

Path Stability & Transparency. Any bandwidth-reservation system requires the source to determine which ASes are traversed to obtain and use the correct reservations, and the paths need to remain stable through the duration of the reservation. Therefore, we assume that the source is aware of the paths to the destination and their validity periods to be able to set up appropriate reservations. This assumption is fulfilled, e.g., by path-aware network architectures that carry the path of packets in their headers such as SCION. As SCION is deployed commercially, with over a dozen ISPs selling connectivity[2], we build Hummingbird on SCION.

PKI for Autonomous Systems. To ensure the validity of reservations, the control plane assumes that a PKI for ASes is in place, such as the Resource Public Key Infrastructure (RPKI) [35] used in secure BGP, or the Control-Plane PKI (CP-PKI) [9] used in SCION.

Allocation of AS-internet resources. We assume that ASes can allocate internal resources to support all reservations that they provide, i.e., traffic can pass between any two interfaces without causing congestion, as long as the advertised available bandwidth per interface is not exceeded.

Time Synchronization. To check the age of a packet, the clocks of hosts using reservations and ASes must be synchronized to a maximum clock skew of δ (e.g., 0.5 seconds). Several time synchronization systems amply satisfy this property, such as NTP, PTP, GNSS – in practice, time accuracy with NTP is around 10ms, and below 100ms in the vast majority of cases [50]. A time synchronization error above 0.5s can invalidate the QoS reservation.

AS Stack. ASes that provide reservations need to deploy a Hummingbird Service, which is responsible for coordinating the reservations and managing the AS-local secret keys for data plane authentication. Further, border routers need to be augmented to support the key derivation and packet authentication processes (see Section 4.3).

Client Stack. Hummingbird clients must be able to (i) contact the control plane to obtain the reservation authentication keys, and (ii) use the keys to authenticate packets at forwarding time.

3.3 High-Level Design

In the following, we provide an overview on how Hummingbird works, illustrated using Fig. 1, and how it provides the properties described in Section 3.1.

At a high level, a flyover reservation is a six-tuple indicating which AS granted the reservation, the ingress and egress interfaces traffic will traverse, the start and expiration times, and the amount of forwarding bandwidth reserved.

Obtaining Reservations. To keep Hummingbird's operation simple and lower the burden on transit ASes, the source is responsible for obtaining valid overlapping reservations that cover the communication path on the control plane.

[2]see https://www.scion.org/isps/

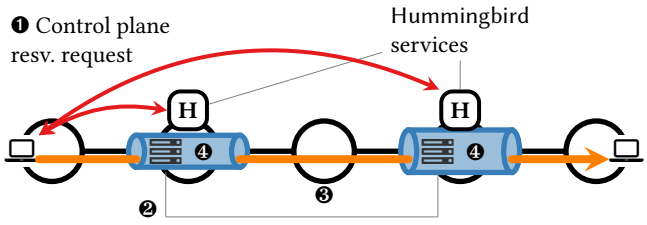

Figure 1: Overview of Hummingbird's operation on a path of five ASes (dark circles). The source host performs a reservation request (❶) to set up reservations on the path to the destination. Reservations may have independent sizes, start and expiration times (represented by the "tube" widths, ❷). Further, not all on-path ASes *must* provide reservations (❸). Border routers (❹) authenticate and prioritize reservation traffic on the data plane.

The control plane coordinates the creation of reservations, and allows establishing a shared reservation authentication key between the source and the transit AS. Since reservations are granted on a per-hop basis, the control plane only requires pairwise interaction between the source and each individual on-path AS (❶ in Fig. 1), and avoids complex multi-party reservation protocols.

In Hummingbird, reservations have a *start time*. Thus, the reservation key can be provided to the source ahead of time, ensuring that the reservation is immediately usable, without any setup delays after the start time.

Since reservations are independent for each hop, reservations covering multiple hops do not need to have the same start time, expiration time, or even bandwidth (❷ in Fig. 1). Further, thanks to this decoupling, the source can obtain partial reservations on the path (❸ in Fig. 1), even if not all on-path ASes provide reservations.

Our control plane provides incentivization and efficient utilization of resources by enabling *bandwidth markets*, in which ASes can sell their reservation bandwidth by listing *bandwidth assets*—which can be split, combined, and traded—that act as vouchers for reservations on the data plane. Once a source wants to use a reservation, it *redeems* the asset to obtain the corresponding reservation authentication key.

In the bandwidth-reservation setting, where decentralization and resilience to failures are of paramount importance, a centralized bandwidth marketplace is unsuitable because it introduces a single point of failure. Because of this, we design a control plane based on blockchain smart contracts that is decentralized, provides strong availability guarantees, and has a high performance. This approach provides multiple advantages and is well-suited to the inherently decentralized setting of the Internet where not all parties mutually trust each other [55]. Blockchains are designed to be Byzantine fault tolerant (BFT) and thus highly resilient to failures and adversarial actions. They are replicated among globally distributed entities and thus provide redundant access to the Hummingbird control plane.

The use of smart contracts allows for flexible listing, modification, and purchase of assets with guaranteed integrity and without the interaction of the selling AS (see Section 4.2). Further, smart contracts are executed atomically, i.e., state changes are only applied if the

whole contract call executes without error, which enables sources to purchase reservations for a whole path in a single transaction, directly solving the problem of providing atomic path reservations.

It is important to note here that modern high-performance blockchains such as Sui [6], which we use for our evaluation, do not suffer from often cited disadvantages that have plagued previous blockchain generations. They are far faster, cheaper, and much more energy efficient.

Using Reservations. At forwarding time, the host uses the reservation keys obtained on the control plane to derive a per-packet message-authentication code (MAC) for each reserved AS hop. Border routers in transit ASes can re-derive this reservation key on the fly based on the reservation information in the packet (❹ in Fig. 1). These cryptographic tags authenticate the reservation information and the packet's length, so that the reserved bandwidth can only be used by the source (see security analysis in Section 5).

Further, reservations are not tied to flows. A source can use the same reservation for multiple flows crossing the same AS hop – while ensuring that the flows *in the aggregate* stay within the reserved bandwidth. Hummingbird's design allows policing reservation traffic to be lightweight, requiring only a single 8 byte counter per reservation (see Section 4.4).

4 System Details

This section presents the details of Hummingbird. We first explain how reservations in our system are authenticated, then provide an example control plane that makes use of smart contracts and allows trading bandwidth assets in a market, and finally discuss the details of the data plane.

A full specification of the packet headers and format, based on the SCION Internet architecture [9], is included in Appendix A. In our specification, traffic that is (partially) protected by reservations uses a separate SCION path type that introduces additional 8 bytes per reserved hop compared to the standard SCION header. We chose SCION as a basis for Hummingbird as it provides path transparency (an end host knows and can observe and select paths) and path stability (the packet's embedded path is followed), which are essential properties for bandwidth-reservation systems (see Section 3.2). Moreover, SCION is deployed on commercial networks, enabling a roll-out of Hummingbird in practice.

While SCION offers native support for path awareness and cryptographically-enforced stability, the core of Hummingbird's design can be adapted to other architectures as well. For instance, IPv6 Segment Routing (SRv6) [15] or MPLS-based [48] networks can support path transparency, and thus would also allow Hummingbird reservations. However, these architectures also have limitations, compared to SCION, that make them less suitable for Hummingbird. SRv6 leverages IPv6 addresses, which are routed using BGP, and thus it is susceptible to the same transient rerouting events and hijacking attacks that prevent path stability in the current Internet. MPLS-based networks, on the other hand, provide stable paths (unless the MPLS labels are announced and discovered through, e.g., BGP-LU [43]), but the configuration of MPLS network-to-network interfaces (NNIs) is still a complex and mostly manual process, limiting the scalability and flexibility of the system. Thus, SCION remains the most practical target at present for realizing end-to-end enforceable QoS with market-based reservations.

4.1 Reservation Authentication

Assume that the source has reserved BW bandwidth from the ingress interface In to the egress interface Eg of an AS K, from time $StrT$ for a duration Dur. AS K assigns an identifier, $ResID$, to the reservation, which must be unique for the interface pair during the reservation's validity period.

The reservation is then determined by the tuple

$$ResInfo_K = (In, Eg, ResID, BW, StrT, Dur). \tag{1}$$

The AS K providing the reservation is implicitly specified by the authentication key for the reservation, which is computed by AS K as

$$A_K = \mathsf{PRF}_{SV_K}\left(ResInfo_K\right), \tag{2}$$

where SV_K is a secret value known only to AS K, which is shared among its border routers. PRF is a secure pseudo-random function with an output length sufficient to yield secure symmetric cryptographic keys (i.e., in practice at least 128 bits). The authentication key, A_K, is shared through the control plane (see Section 4.2) with the source.

Note that, in contrast to previous bandwidth-reservation solutions, no source address or unique source identifier is used to create the reservation in our proposal. Instead, a unique $ResID$ identifies the reservation at each hop. This approach has several advantages:

First, each AS K independently determines the $ResID$, which only has to be unique for a particular interface pair, and thus controls the maximum number of $ResID$s available. This can improve the efficiency of monitoring and policing, which can be done deterministically (see Section 4.4).

Second, reservations can be shared between multiple (mutually trusting) sources or it can be obtained by one party for use by another. For example, a client could obtain a reservation for the reverse direction (i.e., server to client) and provide the authentication key to the server in order to obtain bi-directional bandwidth guarantees (see Appendix C).

Further, the same source may have multiple reservations on the same hop. This can increase the availability guarantees of the system (importantly, it can prevent attacks from on-reservation-set adversaries, see Section 5.1). It also increases the flexibility of the system, e.g., during bursts of traffic, a source can obtain additional reservations to supplement an existing long-standing reservation.

Finally, it enables control-plane-independent reservations without introducing attacks that may be present if the reservations are granted per-source based on network identities. If network identities would be used instead, the source would need to prove its identity when obtaining the reservation (e.g., through a PKI). Otherwise, an adversary could reserve bandwidth on the source's behalf to prevent that source from creating further reservations.

4.2 Control Plane

The Hummingbird data plane is compatible with any control plane that allows the source to (i) negotiate reservations and (ii) obtain

$ResInfo_K$ and A_K from AS K, importantly without relying on in-band identities. This control-plane independence enables the creation of a market for bandwidth reservations, which maximizes the utility of the allocated resources instead of relying on an in-band control plane.

Our control plane enables ASes to represent bandwidth as assets on a blockchain, makes them freely tradable, and allows obtaining end-to-end guarantees atomically. The core of our control plane is an *asset contract* that defines the structure and behavior of *bandwidth assets*. These bandwidth assets are issued by ASes that provide Hummingbird reservations and act as vouchers for said reservations on the data plane. Each asset represents reserved bandwidth for an ingress or egress interface at the issuing AS during a given time interval. A pair of an ingress and an egress asset can then be redeemed at a later time for the information required to use the reservation on the data plane.

The asset contract is complemented by one or multiple independent *market contracts* which provide a decentralized marketplace for trading bandwidth assets. Assets can be split in the time or bandwidth dimension into multiple non-overlapping assets. This allows an AS to issue a single "large" asset that represents all available bandwidth for a given interface for a long time interval, which can then be split by their current owner—which can be the AS, a buyer, or the market contract— into "smaller" assets as needed.

In the following, we will first describe the structure of a bandwidth asset, and then explain the registration process for ASes, before we detail how an end host obtains a reservation. Figure 2 illustrates this process in an example, in which we assume that the AS has previously registered with the asset contract, and that the end host already owns an ingress asset for the desired flyover reservation (❶), e.g., from a previous purchase.

Asset Representation. As reservations should be possible for both long and short time periods as well as for large and small bandwidth allocations, issuing an individual asset for each possible reservation quickly becomes infeasible. Because of this, we represent reservations using assets that are splittable in both the time and bandwidth dimension, subject to the limitations of the provided granularities (see below). This allows an AS to publish one large bandwidth asset, which can then be split up into smaller bandwidth assets as needed, which will be explained below.

Additionally, providing individual assets for all possible pairs of ingress/egress interfaces can quickly lead to a space explosion. We avoid this by making each asset represent a reservation on a single interface used either as ingress or egress for the reservation. To later make use of the reservation, an asset for both an ingress and egress interface from the same AS representing the same bandwidth and time intervals are needed.

Each bandwidth asset contains the following attributes:

AS Identifier: The identifier of the AS offering the reservation. This is set automatically after authenticating the AS.

Bandwidth: The bandwidth of the reservation. Corresponds to *BW* on the data plane.

Start Time: The start time of the reservation represented by the asset. Corresponds to *StrT* on the data plane.

Expiration Time: The expiration time of the reservation represented by the asset. Corresponds to *StrT* + *Dur* on the data plane.

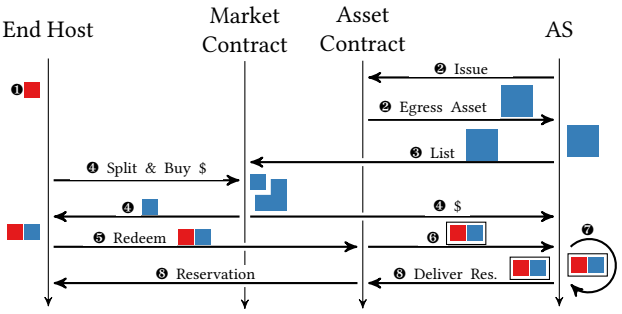

Figure 2: Control-plane interactions between the asset and market smart contracts, an AS, and an end host. The figure shows how assets are issued (❷) and listed on a marketplace (❸), how an end host obtains an egress asset (❹) that matches a previously obtained ingress asset (❶), and how the reservation is redeemed (❺–❽).

Interface: The ID of the AS interface for which the reservation is valid. Corresponds to *In* or *Eg* on the data plane.

Ingress/Egress: An indicator describing whether the reservation allows the use of the interface as ingress or egress.

Time Granularity: The minimum duration for which the AS wants to support reservations. Supporting very short reservations may not be in the interest of every AS, thus we allow each AS to set this threshold. Each asset can be split in the time dimension as long as the duration of each new asset is a multiple of this granularity.

Minimum Bandwidth: The minimum bandwidth of reservations. This parameter allows an AS to limit the number of concurrent reservations, see also Section 4.4. Each asset can be split in the bandwidth dimension as long as the bandwidth of each new asset exceeds this minimum.

AS Registration. Our asset smart contract requires a registration process, which ensures that the entity issuing an asset is actually authorized to do so; i.e., an entity cannot simply create an on-chain asset that represents a reservation at some AS that is not under their control. Since we assume that a PKI for ASes is in place, each AS could simply sign each asset that they issue, which would allow any user to verify the validity of an asset before obtaining it.

This naive approach can be improved to only require authentication once, during an AS registration process, in which the AS provides its AS certificate to the asset smart contract and proves that it is in possession of the corresponding private key. At this point, the smart contract issues an authorization token to the AS that contains the AS identifier. The asset contract then enforces that (i) only the owner of such an authorization token, i.e., registered ASes, can issue bandwidth assets, and (ii) that the AS identifier listed in each issued asset corresponds to the AS identifier contained in the authorization token that is provided during issuance.

Issuance. Each registered AS can issue bandwidth assets for ingress and egress interfaces by calling the *issue* function of the asset contract, specifying the attributes of the asset. In Step ❷ in Fig. 2, the AS issues a large bandwidth asset for an egress interface, which the contract immediately transfers to the AS. At this point, the asset is

owned by the AS and can be transferred arbitrarily. In our example, the AS now lists this asset for sale by sending it to a market contract (❸), which allows the market contract to modify the asset, governed by the restrictions put in both the market and asset contracts, e.g., dictating how assets can be split.

Asset Splitting & Purchase. In our example, the AS has issued, and listed for sale, an asset that represents a longer time interval and larger bandwidth than the end host wants to purchase, namely, an asset that matches the dimensions of the previously obtained ingress asset (❶).

Luckily, the owner of a bandwidth asset—which, in the example is the market contract—can split it in the time or bandwidth dimension if the split conforms to the restrictions on the time granularity and minimum bandwidth, resulting in two "smaller" assets. When splitting an asset, the attributes Start/Expiration Time and Bandwidth are adjusted according to the split, whereas the other attributes remain unchanged. To split an asset into multiple chunks or in both dimensions, the process is repeated.

Thus, in our example (Fig. 2), the end host can buy a fraction of the asset by sending the required payment to the market contract and specifying the desired time interval and bandwidth (❹). The payment is forwarded to the seller (in this case the AS) and the asset that was split off is transferred to the end host, who now becomes its owner. This allows them to transfer or resell the asset, to split it again, or to fuse it with other compatible assets.

Asset Redemption. Before a reservation can be used on the data plane, a pair of ingress/egress assets that represent this reservation need to be *redeemed*. In this step, the owner of said assets exchanges them for the required data-plane information. This is necessary since ownership of assets themselves cannot directly allow use of the network resource, as this requires information (in particular the authentication key A_K) that (i) needs to remain secret, (ii) depends on attributes that change with each split of the asset, and (iii) depends on both assets that are combined for the reservation.

Once an end host has obtained compatible ingress and egress assets (i.e., same AS, validity period, bandwidth) and wants to use the reservation, the end host—as owner of the assets—sends both assets, as well as an ephemeral public key to the asset contract to redeem them (❺). The contract wraps the assets and the public key into a *redeem request* (represented as another on-chain asset) which is then transferred to the issuing AS (❻). Once the AS receives this redeem request, it creates the information required for the reservation (i.e., $ResInfo_K, A_K$) as described in Section 4.1 and encrypts it with the public key from the redeem request (❼).

In this step, the AS also assigns the reservation's *ResID*. To keep monitoring as efficient as possible (see Section 4.4), per-ingress unique IDs are preferred; however, an AS is free to assign IDs differently as long as it can uniquely identify and monitor a reservation based only on the *ResInfo*.

The AS then sends the encrypted reservation (together with the redeem request asset) back to the asset contract, from which it is received by the client (❽). The contract additionally destroys the bandwidth assets, which can thus no longer be traded. Finally, the client decrypts the received information and forwards $ResInfo_K$ and A_K to its Hummingbird-enabled local applications to use on the data plane.

Atomic End-to-End Guarantees. One crucial property we get from using blockchain-based assets is that multiple transactions can be made atomic, i.e., they either all succeed or all fail. Thus, even though each asset represents only an interface of a hop on the path, a source can create an atomic transaction to buy reservations for all hops on a path: If it succeeds, the source receives an end-to-end bandwidth reservation; if it fails, no money is lost (except minimal transaction fees).

The atomicity of blockchain transactions solves one of the most difficult tasks in hop-based bandwidth-reservation systems: Obtaining end-to-end reservations without active coordination between the involved ASes and with no significant partial-failure costs.

4.3 Data Plane

The data plane for Hummingbird flyovers is lightweight, allows stateless traffic authentication on routers, and only requires a small state for policing (see Section 4.4).

When sending a packet over the reservation, after the start time has passed, the source authenticates the packet with the *packet authentication tag*

$$V_K = \mathsf{PRF}_{A_K}(DstAddr, \mathrm{len}(pkt), TS)\ [:\ell_{\mathrm{tag}}], \tag{3}$$

where $\mathrm{len}(pkt)$ is the length of the packet (including header) and TS is a high-resolution timestamp that is unique for each packet, used for traffic policing. *DstAddr* is the destination address, which is included to mitigate reservation stealing attacks (see Section 5) and $[:\ell_{\mathrm{tag}}]$ denotes truncation to ℓ_{tag} bytes, which is a parameter configured at the protocol level. Truncating the tags reduces the effort required for brute-force attacks. However, since the tags are only valid for a short amount of time (see below), and brute-forcing them requires an online attack, this is sufficient for our use case (see Section 5) and allows saving valuable space in the packet headers. In systems like SCION [9], where each hop already contains a *Hop Field MAC* to authenticate the forwarding information, V_K can be aggregated with this Hop Field MAC into an aggregate MAC [26] by XORing the two, which saves additional space (see Appendix A.4).

The packet is then transmitted, containing the reservation information (1) and tag (3) for each hop with a reservation:[3]

$$(TS, \mathrm{len}(pkt), ResInfo_1, V_1, \ldots, ResInfo_K, V_K, \ldots). \tag{4}$$

Upon receiving a packet, each border router performs the following steps:

(1) Check if the packet contains a reservation. If not, treat it as best-effort traffic.
(2) Verify the authenticity of the reservation information in the packet to ensure that the packet is using bandwidth of a valid reservation. The border router only needs to recompute A_K and V_K, and compare the latter to the one in the packet to verify the validity of the reservation. If the authentication fails, the packet is dropped.
(3) Check if the timestamp TS in the packet is recent—i.e., between $t - \delta - \Delta$ and $t + \delta$, where t is the current time, δ is the maximum clock skew, and Δ is the maximum packet age—and that it falls within the validity period of the reservation. If this is not the

[3] Parts of the *ResInfo* are already contained in the SCION standard header.

case, treat the packet as best-effort traffic. Otherwise continue with the next step.
(4) Monitor the reservation traffic, see Section 4.4.
(5) Forward the packet with priority if there is still sufficient bandwidth available in the reservation, otherwise forward with best effort. Packets are not dropped to ensure that benign bursts of traffic—which can occur through network elements that are not under the control of the source—do not degrade performance to below best-effort performance.

Since reservation traffic is prioritized over best-effort traffic, it will be able to pass, even if the network is congested. However, unused bandwidth from a reservation is not wasted, since it is still available for best-effort traffic.

4.4 Traffic Policing

Hummingbird is designed to allow every AS to perform deterministic monitoring and policing for its own reservations to ensure that it can provide its guarantees. This is in contrast to systems like Colibri [17], where the source AS performs deterministic traffic policing and on-path ASes only perform probabilistic monitoring.

For each reservation, policing can be implemented with a simple token-bucket algorithm using a single 8 B counter, as described below. Since an assigned *ResID* is unique for the particular ingress interface during the reservation's validity period, an AS can simply keep an array of $ResID_{\max}$ counters, where $ResID_{\max}$ is the highest assigned *ResID* of all its reservations, and use the *ResID* in the packet header as an index. If the *ResID*s are assigned such that $ResID_{\max}$ is sufficiently small, the policing array can fit into a processor's cache, making policing very efficient.

ResID Assignment. Assigning *ResID*s is an instance of the online interval coloring problem which can be solved efficiently [21, 25, 28, 29]. However, online graph coloring algorithms cannot assign the optimal coloring and instead, in the worst case, assign a maximum color $R \cdot \chi$, where χ is the chromatic number of the graph and R is called the *competitiveness* of the coloring algorithm. In our case, the competitiveness is the ratio between the largest assigned *ResID* and the maximum number of concurrent reservations (i.e., $TotalBW/MinBW$).

By setting the total bandwidth *TotalBW* available for reservations during a time period as well as the minimum bandwidth *MinBW* for reservations, an AS can ensure that the possible values for the *ResID*s are between 0 and $ResID_{\max} := R \cdot TotalBW/MinBW$ in the worst case. An AS can always ensure that this array fits into the processor's cache by choosing appropriate values for *TotalBW* and *MinBW*. Consider the following two examples (using $R = 3$):[4]

(1) An ingress interface with a total bandwidth of 100 Gbps (a common link size in the current Internet [39]) and a min bandwidth of 100 kbps (sufficient for VoIP [46]): This results in $ResID_{\max} = 3 \times 10^6$ and a policing array of 24 MB, which fits into the L3 cache of many current processors.
(2) The same total bandwidth but a minimum of 4 Mbps, which is sufficient for 1080p video calls [46]: This results in $ResID_{\max} = 75\,000$ and a 600 kB policing array, which fits into the L2 cache of current processors.

[4] In the optimal online algorithm $R = 3$ [29], although other algorithms perform better in practice for most interval graphs [21].

Using the *ResID* directly as an index into the policing array and keeping the reservation bandwidth in the packet header makes the policing algorithm extremely simple.

Traffic Policing Algorithm. Algorithm 1 describes a simple and highly efficient algorithm, adapted from the algorithm presented by Wyss et al. [54], for deterministic policing, which only keeps an 8-byte timestamp per reservation in an array plus a global parameter *BurstTime*.

This parameter limits the allowed "burstiness": intuitively, a sender cannot send more than *BurstTime* worth of traffic compressed into a single burst. It should be set sufficiently small such that the router's buffers can absorb some synchronized bursts and to limit jitter, but large enough that it does not cause unnecessary constraints on packet sizes for small reservations.

Given current trends in router buffers [58], a value of roughly 50 ms seems reasonable. Note that for very small reservations (below 240 kbps), this limits the maximum packet size to below 1500 B. However, specifically for VoIP applications, this is not an issue, as those anyway send a packet roughly every 20 ms [10] and thus do not trigger the burst prevention.

Processing a packet requires very few operations, namely i) getting the current time, ii) 1 load operation, iii) 2 comparisons and branches, iv) 1 division[5], v) 2 additions, and vi) 1 store operation (in the successful case).

Algorithm 1 Simple policing algorithm for Hummingbird which assumes unique values for the *ResID* of all reservations on a given ingress interface. This algorithm locally stores an array of timestamps (one for each possible value of *ResID*) and a global parameter *BurstTime*. All other inputs (*ResID*, *BW*, *PktLen*) are part of the authenticated packet information. The algorithm is a slight adaptation of the algorithm presented in Appendix E of Wyss et al. [54].

Input: *TSArray*, *BurstTime* (local store)
Input: *ResID*, *BW*, *PktLen* (packet)
Returns: fwd_type

```
function BandwidthMonitoring(ResID, BW, PktLen)
    now ← time()
    TS ← max(TSArray[ResID], now) + PktLen/BW
    if TS ≤ now + BurstTime then
        TSArray[ResID] ← TS
        Return: fwd_flyover
    else
        Return: fwd_best_effort
```

5 Security Analysis

In this section, we analyze the security of Hummingbird. We first introduce the adversary model, then describe the security properties, and finally analyze the security of the control and data planes.

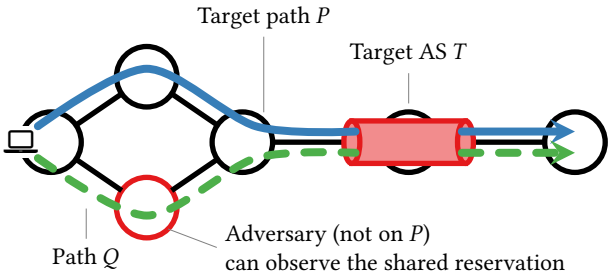

Figure 3: On-reservation-set adversary. The source is using two paths P (solid line) and Q (dashed line) concurrently, with a single shared reservation on AS T. Despite not being on the target path P, the adversary can observe authentication tags that are also valid for path P, since the same reservation is used for path Q.

5.1 Adversary Model

To analyze Hummingbird's security, we consider an adversary that controls a set of hosts, ASes, and/or routing elements. The adversary is distributed and can drop, modify, duplicate, inject, or delay traffic but not break cryptography. We further make the following assumptions to allow a meaningful analysis of the system's properties:

(1) There exists a routable path of honest ASes from source to destination. Otherwise, no availability guarantees can be provided in any system since the adversary can always drop the traffic and prevent any communication.
(2) Every honest source and AS is able to communicate with the control-plane elements to obtain reservations. For our control plane, this means that each source and each AS must have access to a path of honest ASes that connects them to the blockchain network on which the control-plane smart contract is hosted. In other words, we assume that the adversary cannot eclipse [22, 52] an honest party.
(3) There is an efficient (in the economic sense) market that allows bandwidth-reservation assets to be priced correctly.

We distinguish several different adversaries. Considering a source S and a destination D communicating over a path P, we distinguish the following adversary types:

Compromised source: The adversary tries to gain an unfair advantage as the source of the communication.

On-path adversary: The adversary controls at least one AS or routing element on the path P (but not the source S).

On-reservation-set adversary: This type of adversary is illustrated in Fig. 3, and can only exist if the source simultaneously uses two paths P and Q that share a reservation on at least one target hop. The adversary then controls at least one AS that is on path Q but not on P and targets reservation traffic on path P. Thus, this adversary can observe valid packet authentication tags for the reserved downstream ASes that are shared between paths Q and P.

[5]If desired, the division in line 3 of Algorithm 1 can be replaced by a load and multiplication by storing the inverse of all possible 1024 BW values in an array (assuming a 10-bit field as described in Appendix A.4).

Off-path adversary: The adversary controls a set of hosts and ASes, but is not one of the previously defined types.

5.2 Security Properties

Hummingbird provides the following security properties, on the control plane (C1, C2) and data plane (D1, D2):

C1 *Secure reservation establishment.* The control plane allows an honest host to securely establish a reservation, i.e., the control-plane guarantees confidentiality for the reservation authentication keys, and integrity of the reservation information. In addition, the adversary cannot perform *denial-of-capability attacks* that prevent an honest host from obtaining reservations on an honest AS hop—provided that there is sufficient bandwidth available.

C2 *Fair bandwidth distribution.* The adversary cannot gain more than a fair share of the bandwidth available for reservations, by, e.g., creating multiple accounts or using large numbers of hosts (Sybil attack). This abstract property depends on the specific notion of fairness. In this paper, we consider the notion of *economic fairness*: An adversary cannot obtain a larger reservation than other nodes without paying the fair market price for it. Equivalently, the adversary cannot starve honest nodes for bandwidth without paying the respective price.

D1 *Overuse protection.* An adversary cannot overuse or spoof a reservation. The adversary cannot use a forged reservation, or a valid reservation (at an honest AS hop) outside its limits, i.e., before start time, after end time, or for more bandwidth than granted.

D2 *Quality of Service (QoS).* Legitimate hosts should benefit from the QoS guarantees promised by reservations at an honest AS. These guarantees include the amount of bandwidth reserved, but may also incorporate, e.g., bounds on jitter, latency, or other network-layer properties that may be provided by the AS inside their domain. As an on-path adversary can always perform a DoS attack, and thus disrupt QoS, by simply dropping packets (as in all bandwidth-reservation systems), we do not consider this adversary type in our analysis of this property.

5.3 Control-Plane Analysis

Secure reservation establishment (C1). Secure reservation establishment is ensured through our assumptions on the control plane in combination with the registration of ASes and the encryption of $ResInfo_K, A_K$ during redemption.

Since we assume that a PKI is in place for ASes (see Section 3.2) and ASes prove to the asset smart contract that they are in possession of the secret key corresponding to their certificate, each issued asset provides authenticity. Due to the access-control guarantees provided by the blockchain and the smart contracts (only the owner of an asset can use the asset in a transaction), no other party can tamper with the message sent to deliver the reservation, which ensures its integrity. Confidentiality is provided by encrypting the asset with the public key of the asset owner (i.e., the host). This public key is sent to the AS together with the asset through the smart contract, which guarantees authenticity of the key (again, since only the owner can use the asset) and therefore confidentiality of messages encrypted with this key.

Finally, our assumption that honest hosts and ASes are not eclipsed (Section 5.1) and can therefore always reach the blockchain hosting the smart contracts ensures the availability of the marketplace, and protects against attacks that prevent an honest host from obtaining a reservation.

Fair bandwidth distribution (C2). Fairness and Sybil resistance are guaranteed by allowing the assets to be freely tradeable and our assumption that an economically efficient market exists that enables price discovery for a fair market value. The existence of such a market then guarantees *economic fairness*. Even if an adversary was able to obtain reservations at a price that is lower than the value of the reservations at the time of an attack (e.g., by buying the reservations far in advance when they are valued lower), they would suffer the opportunity cost of not reselling the bandwidth assets. Thus, the cost of the attack is always at least equal to the value of the corresponding reservation. The same applies to starving honest sources of bandwidth: To do so, the adversary would have to reserve the full hop bandwidth themselves—and therefore pay for it—becoming its legitimate owner. Note that the adversary gains nothing from creating multiple hosts or accounts, i.e., a Sybil attack, since they need to pay the market value for each asset, independent of the number of accounts that they are using to obtain reservations.

A concrete market design or analysis thereof is out-of-scope for this work. However, there are several well-known market mechanisms that could plausibly be deployed to achieve efficient price discovery and allocation of bandwidth reservations. One possibility are Vickrey-Clarke-Groves (VCG) auctions [11, 19, 47], in which participants bid on multiple items simultaneously and the final allocation maximizes the total utility. These auctions are provably strategy-proof and economically efficient under a broad set of assumptions and could be utilized to vote for bandwidth reservations along a path. However, this would require additional rounds of communication with a smart contract as well as discrete rounds in which the auctions complete.

Alternatively, a more lightweight approach could involve a spot market with posted prices, similar to our prototype, where Autonomous Systems (ASes) list bandwidth offers in real time, and users select the best-priced paths that satisfy their requirements. Such markets offer responsiveness and deployability, albeit with potential inefficiencies under high volatility or demand shocks. Hybrid mechanisms, e.g., combining advance reservation auctions with short-term spot market resales, could help stabilize pricing while maintaining flexibility for bursty traffic.

Crucially, the path diversity in the underlying SCION architecture enhances the viability and liquidity of any such market. Between most source-destination pairs, there are multiple disjoint or partially overlapping paths, recent measurements in SCIERA, an educational network that is part of the production SCION network, show that between most source/destination pairs, there are more than twenty, and up to one hundred, paths available, and at least two paths exist between any pair [14]. This diversity provides many alternative paths that constitute substitutable goods in economic terms: if one path segment is overpriced or congested, alternative paths can be used instead. This not only reduces monopolistic pricing risks, but also improves price stability and makes the market more resilient to manipulation or scarcity-induced spikes.

We argue that the design of Hummingbird enables a range of plausible mechanisms that are capable of achieving sufficient price discovery and economic fairness to support our assumption. Future work may explore these mechanisms in more detail, potentially adapting tools from existing pricing research and auction theory to the unique constraints of inter-domain bandwidth reservations.

5.4 Data-Plane Analysis

Overuse protection (D1). Hummingbird protects against reservation overuse and spoofing through its per-packet reservation authentication and traffic policing. The authentication tag is computed over the full reservation description and a secret value only known to the granting AS, and each AS performs traffic policing based on these protected values. Therefore, an adversary that either tries to spoof a reservation or to overuse a valid reservation must be able to forge an authentication tag. Since we assume that the authentication tags are computed using a secure PRF, the authentication tags are a secure MAC with security parameter ℓ_{tag}.

This ensures that the only possible attack vector is a brute-force attack on the authentication tag, the feasibility of which depends on their length ℓ_{tag}. To save space in the packet header, tags are ideally as short as possible, while providing sufficient security in practice.

A brute-force attack on Hummingbird cannot be performed offline, as it is not possible to check the validity of a tag without knowledge of the key. This means that for each candidate tag, a packet needs to be sent to the reservation AS, which forces the adversary to brute-force the tag in an online attack. Brute-forcing a valid tag only allows an adversary to use it for a short time, namely the validity period (e.g., 1 second) of a packet with the provided timestamp. Therefore, the tag length ℓ_{tag} only needs to be chosen such that an online attack becomes infeasible (or prohibitively expensive) for an adversary. In our implementation, we use a tag length of 6 bytes, which would require the adversary to send (on average) more than 140 trillion packets to have one success, which is prohibitively expensive in practice. Note that brute-forcing the key used for the PRF—which would allow sending packets until the expiration time of the brute forced reservation—depends on the key length, not the tag length, and is therefore infeasible.

Quality of Service (D2). In an honest AS, QoS is provided by prioritizing the reservation over best-effort traffic, ensuring that the source can use the full reservation bandwidth.

An adversary can try to cause delayed or dropped packets, e.g., by introducing congestion at the reserved hop. Since an honest AS does not hand out more reservations than it can handle and prioritizes reservation traffic, congestion caused by best-effort traffic does not impact reservations.

This does not exclude the threat of other DoS attacks, which we analyze separately for off-path and on-reservation-set adversaries. As discussed above, brute-force attacks on authentication tags are not feasible in practice. Therefore, off-path adversaries cannot forge authentication tags for an honest AS hop, which prevents them from disturbing reservation traffic, since they can only either send best-effort traffic, or traffic belonging to a valid reservation.

Since reservations are granted for individual hops, a host may use the reservation for a specific hop on multiple paths. In this case,

an on-reservation-set adversary could observe valid authentication tags for reservations on hops that are used by the source for multiple paths. This enables such an adversary to perform a DoS attack (D2) on these hops.

Namely, an on-reservation-set adversary can observe authentication tags for the reserved hop from packets that were sent on a path that includes the adversary. The adversary can then fabricate packets that will successfully pass authentication at the reserved hop by duplicating the observed packet authentication tags (optionally dropping the original packets) which is possible during the validity period of the observed packet (e.g., 1 second). Since Hummingbird does not require ASes to deploy duplicate suppression, this can be used for a DoS attack on packets that do not flow through the attacker: by sending large amounts of traffic on the reservation to exhaust the reservation's bandwidth limit at a downstream AS, causing legitimate traffic arriving on a different path at the downstream AS to get dropped by the policing system.

To prevent such attacks, it is sufficient to make a separate reservation for shared hops on each path and thus use each reservation exclusively on one path at a time. Whether to use a single reservation on multiple paths or to obtain separate reservations for each path is a trade-off between convenience and security. The decision between these options can be made by the source, depending on whether they trust ASes on all paths that they use, or only the ASes on the current path.

Reservation-stealing as incentive. One possible motive for an attacker is *reservation-stealing*. An on-path or on-reservation-set adversary can attempt to use a reservation from another party for themselves by reusing the authentication tag of a valid packet and replacing the payload. The effect of such an attack for the source is the same as any other DoS attack, and our analysis above applies equally. However, reservation-stealing provides an additional incentive, as the compromised AS can benefit from using the reservation.

As a mitigation, the computation of the authentication tag includes the destination address. This ensures that an adversary can only benefit if they either (i) send traffic to the same destination, or (ii) control an additional on-path AS after the reservation that redirects the packet to the new destination.

On the Absence of Duplicate Suppression and Gateways.

Previous bandwidth-reservation designs (Helia [54], Colibri [17]) require duplicate suppression at on-path ASes and gateways at the source AS. We analyze here why Hummingbird can work without these cumbersome subsystems while still achieving similar security and availability properties.

Both duplicate suppression and the gateway are in place mainly to prevent framing attacks, in which the adversary sends a large amount of seemingly legitimate traffic to frame the source AS, and cause repercussions for the source, such as precluding new reservations. These attacks may work in two ways:

- An on-path adversary may duplicate packets and send them at a high rate. These packets will pass authentication, and it is therefore necessary to remove them before they are accounted for in the monitoring.
- A compromised or malicious host who has access to hop authentication keys may relay them outside the AS, allowing an off-path adversary to generate seemingly legitimate traffic. To prevent this, the gateway was introduced such that authentication keys are not accessible to endhosts.
  It is important to note that this problem stems from the fact that previous systems rely on ASes to manage reservations for scalability reasons, i.e., to aggregate the control plane and monitoring tasks at the AS level.

  In our system, these cases are not a problem for two reasons:

(1) There is no penalty for overusing a reservation: in case of overuse, packets are simply dropped. Therefore, framing attacks are never successful.
(2) The scalability of the system does not rely on AS-level aggregation. Therefore, any entity can obtain authentication keys without external mediation from their AS. The gateway is therefore not strictly necessary, as the monitoring is directly imputed to the source host and the AS is not directly responsible.

The only type of attack that would be prevented in Hummingbird by introducing duplicate suppression are reservation-DoS attacks from on-reservation-set adversaries. We believe, however, that the practical advantages of removing duplicate suppression from the system requirements outweigh this downside, in particular, since the same attacks can be prevented by obtaining separate reservations for each used path.

Further, duplicate suppression can be incrementally added (the system is designed to include unique packet IDs that can be used for duplicate suppression) by individual ASes. Note that, for on-path adversaries, the lack of duplicate suppression does not introduce new attack vectors, since an on-path adversary is always able to block traffic from the source.

The gateway's function of enhancing the system's scalability—by multiplexing multiple AS-internal reservations in one single inter-domain bandwidth reservation—is still beneficial, and our system readily supports the implementation of gateways to this end. However, the gateway is not *required*, which makes Hummingbird more flexible and improves the speed at which it can be implemented and deployed in practice.

6 Control-Plane Evaluation

In this section, we describe the implementation of our control plane and evaluate its performance. Additional evaluation results are provided in Appendix B. The most relevant analysis here is the cost and time required to make a reservation on the marketplace. Note that the scalability of Hummingbird's control-plane operations is dictated by the scalability of the blockchain—which is beyond the scope of this paper. However, a recent study [6] shows that the blockchain we make use of supports thousands of transactions per second.

6.1 Implementation

Control-Plane Smart Contracts. We implemented our control plane as a set of smart contracts on top of the Sui blockchain [6] that provide the bandwidth asset functionality, as well as a marketplace that allows buying and selling assets.

Market Client Application. To allow ASes and end hosts to interact with the smart contracts (i.e., the assets and marketplace), we implemented a client application written in Rust. Our application

Table 1: Gas and dollar cost, rounded to two significant figures, of transactions to atomically buy and redeem a full path. The total cost in SUI is computed as `computation cost + storage cost - storage rebate`.

Hops	Computation	Storage (SUI)		Total	
	(SUI)	Cost	Rebate	SUI	USD
1	0.000 75	0.047	0.016	0.031	0.038
2	0.000 75	0.090	0.029	0.062	0.076
4	0.000 75	0.18	0.054	0.12	0.15
8	0.0015	0.35	0.10	0.25	0.30
16	0.0030	0.69	0.20	0.49	0.60

Computation price: 7.5×10^{-7} SUI/unit; storage price: 7.6×10^{-6} SUI/byte; SUI price: 1.221 USD (as of 2024-04-18 14:09 UTC);

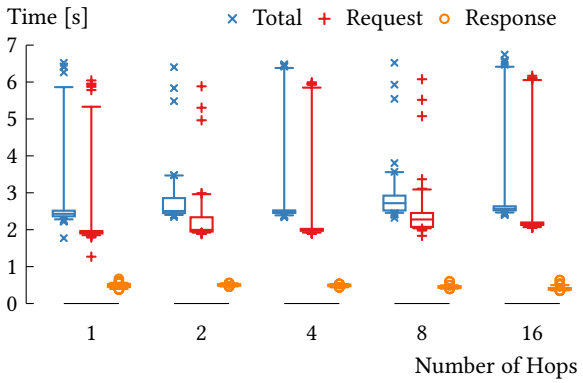

Figure 4: End-to-end latency for an atomic buy-and-redeem (reporting total, request, and response latencies) for different path lengths, each measured 100 times. Time is measured from the point when the buyer has chosen which assets to buy until all of the reservation information required for the data plane has been delivered to the buyer. The whiskers in the box plots represent the 5th and 95th percentiles.

allows an AS to create assets and marketplace listings and handles the assignment of *ResID*s (using an online First Fit algorithm [21, 28]), the derivation of the authentication keys, and the delivery of the reservation information (via smart contract call).

On the end host, the application handles buying and redeeming of assets and integrates with the Hummingbird data plane, i.e., delivers all necessary information to the applications using the reservation.

Blockchain Platform & Atomic Transactions. We chose the Sui blockchain for our implementation due to its high throughput and low latency, which make it suitable for applications such as bandwidth reservations. Sui transactions using only *owned* objects benefit from a fast path (using Byzantine consistent broadcast). This reduces latency compared to accessing *shared* objects, which require consensus. In our design, all operations on the assets themselves can be performed using this fast path. Only interactions with a market require the consensus path.

Evaluation. Our benchmarks measure the cost incurred by the smart contract execution for atomically buying and redeeming reservations for an end-to-end path, as well as the end-to-end latency between initiating the buy-and-redeem transaction, until all reservations are delivered. We perform the latency measurements on the Sui `testnet` which is globally replicated with a distribution that closely resembles its main blockchain (Sui `mainnet`).

6.2 Results

Contract Execution Cost. The cost for transactions in Sui is split into three components. The *computation cost* is charged according to fixed buckets of computational units, based on the computation complexity, which are then converted to a value in Sui according to the *computation gas price* provided in the transaction. In our results, the values are provided based on the current reference gas price on mainnet of 7.5×10^{-7} SUI per unit. The *storage cost* of objects is charged based on a *storage gas price*, which is currently 7.6×10^{-6} SUI per byte. The third component is a *storage rebate*, which the transaction sender receives when deleting an object, consisting of 99% of the original price paid for its storage.

Table 1 shows the cost of atomically purchasing paths of different lengths. Since the cost is deterministic, the table does not include any measurement uncertainties. The cost is dominated by the storage cost resulting from splitting an asset and re-listing the pieces that are not bought. In our benchmark, we perform a worst-case split for each asset on the path, consisting of two splits in the time dimension and one in the bandwidth dimension.

The cost depends linearly on the path length, since a longer path involves more assets that are split off from assets listed in the market. The transaction fees are approximately 0.038 USD per hop, resulting in a total cost of 0.60 USD for a path with 16 hops. Compared to transaction fees on, e.g., Ethereum where the average transaction fees are 8.16 USD[6] this is extremely cheap, and compares favorably even to centralized payment providers such as Paypal or Square, which charge a percentage of the price plus a fee of 0.49 and 0.30 USD, respectively [34]. Importantly, most of the transaction fees are later refunded to the ASes during the redemption of the reservation. Thus, the fees can be partially priced in to the asset listings on the marketplace.

End-to-end Latency. Figure 4 shows the end-to-end latency for atomically buying reservations for a full path. The measurements are run 100 times for each path length. The request time describes the latency between initiating the purchase and the finalization of the purchase transaction. The response time measures the latency from that point until the buyer has received the reservation information from all on-path ASes. The responses make use of Sui's fast path, which lowers the latency significantly, whereas the purchase transaction interacts with a shared object (namely the marketplace), which requires going through consensus. The latency is largely independent of the length of the path with a total latency of less than 3 seconds in 83% of our measurements.

[6] 30-day trailing average transaction fee on 2024-04-18, from https://etherscan.io/chart/avg-txfee-usd

This end-to-end latency is comparable to the time required to authorize credit-card payments ("a few seconds" [45]). Thus, any centralized bandwidth market that makes use of credit-card payments—independent of the underlying QoS system—is unlikely to provide a significantly lower latency. For many applications, e.g., video calls, this should even be fast enough for obtaining reservations ad-hoc, e.g., when congestion occurs unexpectedly. However, Hummingbird also allows reservations to be established ahead of time—which we expect to be the common usage—in which case the latency matters less and is more than fast enough for any application.

7 Data-Plane Evaluation

We implement two versions of the Hummingbird data plane. The first extends the open-source SCION implementation in Golang, adding the Hummingbird data plane and providing an API for the control plane. The second is a high-speed variant, which comprises the generation of reservation traffic at the source and the packet validation and forwarding at border routers. We use this latter version to demonstrate Hummingbird's potential for performance and compare it to a SCION-only data plane, constituting our baseline. We provide additional evaluation results in Appendix B.

7.1 Implementation and Test Setup

We implement both SCION and Hummingbird in DPDK [13], which is also used for SCION routers deployed in commercial networks [1]. For the MAC and PRF operations, we use AES-128, taking advantage of Intel's AES-NI [20] hardware instructions. To deterministically police flyover reservations in Hummingbird, we use an array that establishes a direct mapping between the reservation ID and its corresponding 8B token bucket.We initialize the array to store at most 10^5 reservation IDs, requiring 800 kB of memory.

Our testing environment consists of two machines: one off-the-shelf server with an Intel Xeon 2.1 GHz CPU running Ubuntu and executing our implementation, and a Spirent SPT-N4U device. The Spirent machine serves a dual role as a traffic generator during border router traffic validation evaluations, and as a bandwidth monitor when assessing traffic generation at a source. Both machines are interconnected by four bidirectional Ethernet links operating at 40 Gbps each. We assess each of the two implementations independently. When evaluating Hummingbird, we always measure its worst-case performance by assuming the existence of a flyover reservation at every on-path AS.

7.2 Results

Traffic Forwarding. The border router's validation and forwarding performance are shown in Fig. 5. With only 4 cores, our implementation achieves the line rate of 160 Gbps for 1500 B payloads, and scales to send 100 B payloads at line rate using 32 cores. As anticipated, operations such as computing the SCION hop field MAC, authentication key, flyover MAC, and performing the overuse check emerge as the most resource-intensive operations. We find that best-effort SCION packets can be processed in 123 ns, while Hummingbird packets require 308 ns. Although this difference is noticeable, this does not imply a lower throughput in practice, as it can be mitigated by increasing the number of CPU cores.

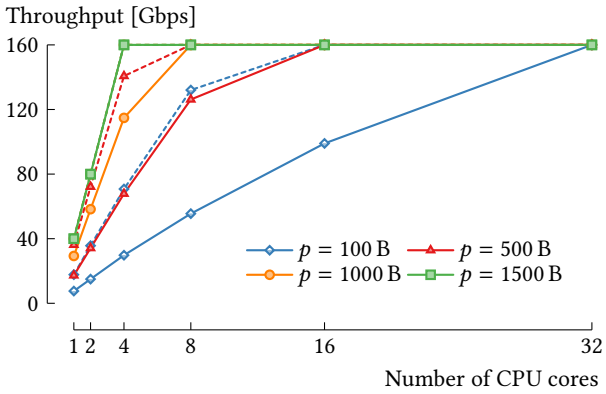

Figure 5: Border router packet validation and forwarding performance for different payload sizes and number of CPU cores. Solid lines correspond to Hummingbird reservations, dashed lines to standard SCION best-effort traffic. Note that the dashed lines for 1000B and 1500B are hidden behind the solid line for 1500B since the performance is virtually identical.

8 Conclusion

We explore how the combination of an out-of-band control plane—in our case in the form of a smart contract running on a blockchain—with a data plane for inter-domain bandwidth reservations can result in a highly flexible, efficient, and fair system to enforce availability guarantees on a public Internet.

The efficient data plane makes Hummingbird easy to deploy on routers. The reservation model—whereby reservations are independently granted on each AS-level hop—facilitates incremental deployment, as not all on-path ASes need to offer bandwidth reservations. Further, the reservation market dynamics incentivize ASes and provide them with an additional source of revenue.

In conclusion, Hummingbird is the first practical inter-domain bandwidth-reservation system: Its substantial benefits for endhosts, and clear economic incentives for ASes facilitate adoption, enabling applications with strong availability requirements to flourish on the SCION Internet.

Ethical Considerations

This paper does not raise any ethical concerns.

Acknowledgments

We gratefully acknowledge support for parts of this project from the Werner Siemens Stiftung Centre for Cyber Trust at ETH Zurich.

References

[1] Anapaya Systems. 2020. SCION-Internet and Anapaya Software. https://content.anapaya.net/hubfs/collateral/anapaya-scion-Internet-and-anapaya-software-fs-en.pdf?hsLang=en.

[2] Tom Anderson, Timothy Roscoe, and David Wetherall. 2004. Preventing Internet denial-of-service with capabilities. *ACM SIGCOMM Computer Communication Review (CCR)* 34, 1 (2004), 39–44. doi:10.1145/972374.972382

[3] Fred Baker, David L. Black, Kathleen Nichols, and Steven L. Blake. 1998. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* RFC 2474. IETF. doi:10.17487/RFC2474

[4] Hitesh Ballani, Paul Francis, and Xinyang Zhang. 2007. A study of prefix hijacking and interception in the Internet. *ACM SIGCOMM Computer Communication Review (CCR)* 37, 4 (2007), 265–276. doi:10.1145/1282380.1282411

[5] Cristina Basescu, Raphael M. Reischuk, Pawel Szalachowski, Adrian Perrig, Yao Zhang, Hsu-Chun Hsiao, Ayumu Kubota, and Jumpei Urakawa. 2016. SIBRA: Scalable Internet Bandwidth Reservation Architecture. In *Symposium on Network and Distributed Systems Security (NDSS)*. The Internet Society, Reston, VA, USA, 16 pages. doi:10.14722/ndss.2016.23132

[6] Sam Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, Brandon William, and Lu Zhang. 2023. *Sui Lutris: A Blockchain Combining Broadcast and Consensus.* Technical Report. Mysten Labs. https://arxiv.org/pdf/2310.18042

[7] Robert T. Braden, Lixia Zhang, Steven Berson, Shai Herzog, and Sugih Jamin. 1997. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification.* RFC 2205. IETF. doi:10.17487/RFC2205

[8] Ignacio Castro, Aurojit Panda, Barath Raghavan, Scott Shenker, and Sergey Gorinsky. 2015. Route bazaar: Automatic interdomain contract negotiation. In *USENIX Workshop on Hot Topics in Operating Systems.*

[9] Laurent Chuat, Markus Legner, David Basin, David Hausheer, Samuel Hitz, Peter Müller, and Adrian Perrig. 2022. *The Complete Guide to SCION.* Springer, Heidelberg, DE. doi:10.1007/978-3-031-05288-0

[10] Cisco. 2016. Voice Packetization. Retrieved August 16, 2023 from https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/voice/cube/configuration/cube-book/cube-book_chapter_01001111.html

[11] Edward H Clarke. 1971. Multipart pricing of public goods. *Public choice* (1971), 17–33.

[12] R. Cocchi, S. Shenker, D. Estrin, and Lixia Zhang. 1993. Pricing in computer networks: motivation, formulation, and example. *IEEE/ACM Transactions on Networking* 1, 6 (1993), 614–627. doi:10.1109/90.266050

[13] DPDK Project. 2023. Data Plane Development Kit. https://dpdk.org.

[14] Wirz et al. 2025. Scaling SCIERA: A Journey Through the Deployment of a Next-generation Network. In *ACM SIGCOMM Conference*. Association for Computing Machinery, New York, NY, USA.

[15] Clarence Filsfils, Pablo Camarillo, John Leddy, Daniel Voyer, Satoru Matsushima, and Zhenbin Li. 2021. Segment Routing over IPv6 (SRv6) Network Programming. RFC 8986. doi:10.17487/RFC8986

[16] Xinzhe Fu and Eytan Modiano. 2021. Learning-NUM: Network Utility Maximization with Unknown Utility Functions and Queueing Delay. In *Proceedings of the Twenty-Second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing* (Shanghai, China) *(MobiHoc '21)*. Association for Computing Machinery, New York, NY, USA, 21–30. doi:10.1145/3466772.3467031

[17] Giacomo Giuliari, Dominik Roos, Marc Wyss, Juan Angel Garcí a Pardo, Markus Legner, and Adrian Perrig. 2021. Colibri: a cooperative lightweight inter-domain bandwidth-reservation infrastructure. In *Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. Association for Computing Machinery, New York, NY, USA, 104–118. doi:10.1145/3485983.3494871

[18] Giacomo Giuliari, Marc Wyss, Markus Legner, and Adrian Perrig. 2021. GMA: A Pareto Optimal Distributed Resource-Allocation Algorithm. In *Proceedings of the International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. doi:10.1007/978-3-030-79527-6_14

[19] Theodore Groves. 1973. Incentives in teams. *Econometrica: Journal of the Econometric Society* (1973), 617–631.

[20] Shay Gueron. 2010. *Intel Advanced Encryption Standard (AES) new instructions set.* Technical Report. Intel Corporation. https://www.intel.com.bo/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf

[21] András Gyárfás and Jenö Lehel. 1988. On-line and first fit colorings of graphs. *Journal of Graph theory* 12, 2 (1988), 217–227.

[22] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse attacks on Bitcoin's peer-to-peer network. In *USENIX Security Symposium (USENIX Security)*. USENIX Association, Berkeley, CA, USA, 129–144. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[23] Hsu-Chun Hsiao, Tiffany Hyun-Jin Kim, Sangjae Yoo, Xin Zhang, Soo Bum Lee, Virgil Gligor, and Adrian Perrig. 2013. STRIDE: Sanctuary Trail – Refuge from Internet DDoS Entrapment.. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. doi:10.1145/2484313.2484367

[24] V. Jacobson. 1988. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.* 18, 4 (Aug. 1988), 314–329. doi:10.1145/52325.52356

[25] Manas Jyoti Kashyop, NS Narayanaswamy, et al. 2020. Dynamic data structures for interval coloring. *Theoretical Computer Science* 838 (2020), 126–142.

[26] Jonathan Katz and Andrew Y Lindell. 2008. Aggregate message authentication codes. In *Cryptographers' Track at the RSA Conference*. Springer, Berlin, Heidelberg, 155–169.

[27] F P Kelly, A K Maulloo, and D K H Tan. 1998. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society* 49, 3 (1998), 237–252. arXiv:https://doi.org/10.1057/palgrave.jors.2600523 doi:10.1057/palgrave.jors.2600523

[28] Hal A Kierstead, David A Smith, and William T Trotter. 2016. First-fit coloring on interval graphs has performance ratio at least 5. *European Journal of Combinatorics* 51 (2016), 236–254.

[29] Henry A Kierstead, William T Trotter, et al. 1981. An extremal problem in recursive combinatorics. *Congressus Numerantium* 33, 143-153 (1981), 98.

[30] Tiffany Hyun-Jin Kim, Cristina Basescu, Limin Jia, Soo Bum Lee, Yih-Chun Hu, and Adrian Perrig. 2014. Lightweight source authentication and path validation. In *ACM SIGCOMM Conference*. Association for Computing Machinery, New York, NY, USA, 271–282. doi:10.1145/2619239.2626323

[31] Soo Bum Lee and Virgil D. Gligor. 2010. FLoc: Dependable link access for legitimate traffic in flooding attacks. In *Proc. of the IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE.

[32] Soo Bum Lee, Min Suk Kang, and Virgil D. Gligor. 2013. CoDef: Collaborative defense against large-scale link-flooding attacks. In *Conference on Emerging Networking Experiments and Technologies (CoNEXT)*.

[33] Taeho Lee, Christos Pappas, Adrian Perrig, Virgil Gligor, and Yih-Chun Hu. 2017. The Case for In-Network Replay Suppression. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. Association for Computing Machinery, New York, NY, USA, 12 pages. doi:10.1145/3052973.3052988

[34] Kimberlee Leonard and Bottorff Cassie. 2023. Credit Card Processing Fees (2024 Guide). Forbes Advisor. Retrieved January 16, 2024 from https://www.forbes.com/advisor/business/credit-card-processing-fees/

[35] Matt Lepinski and Stephen Kent. 2012. *An Infrastructure to Support Secure Internet Routing.* RFC 6480. IETF. doi:10.17487/RFC6480

[36] Jeffrey MacKie-Mason and Hal Varian. 1995. Pricing the Internet. MIT Press.

[37] Michael J. Neely. 2013. Delay-Based Network Utility Maximization. *IEEE/ACM Transactions on Networking* 21, 1 (2013), 41–54. doi:10.1109/TNET.2012.2191157

[38] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce Maggs, and Yih-Chun Hu. 2007. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. In *ACM SIGCOMM Conference*. doi:10.1145/1282380.1282413

[39] PeeringDB. 2023. PeeringDB. Retrieved August 16, 2023 from https://www.peeringdb.com/

[40] Akhil P.T. and Rajesh Sundaresan. 2019. Network utility maximization revisited: Three issues and their resolution. *Performance Evaluation* 136 (2019), 102050. doi:10.1016/j.peva.2019.102050

[41] Verified Market Reports. 2023. Global Managed Leased Line Service Market By Type (Analog Dedicated Line, Digital Line), By Application (BFSI, Medical Insurance), By Geographic Scope And Forecast. Retrieved 2 September 2024 from https://www.verifiedmarketreports.com/product/managed-leased-line-service-market/

[42] Databridge Market Research. 2024. Global Software-Defined Wide Area Network (SD-WAN) Market – Industry Trends and Forecast to 2031. Retrieved 2 September 2024 from https://www.databridgemarketresearch.com/reports/global-software-defined-wide-area-network-sd-wan-market

[43] Eric C. Rosen and Yakov Rekhter. 2001. Carrying Label Information in BGP-4. RFC 3107. doi:10.17487/RFC3107

[44] Benjamin Rothenberger, Dominik Roos, Markus Legner, and Adrian Perrig. 2020. PISKES: Pragmatic Internet-scale key-establishment system. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. Association for Computing Machinery, New York, NY, USA, 73–86. doi:10.1145/3320269.3384743

[45] Stripe. 2023. Card authorisation explained: How it works and what businesses need to know. Retrieved January 17, 2024 from https://stripe.com/en-gb-ch/resources/more/card-authorization-explained

[46] Zoom Support. 2023. Zoom system requirements: Windows, MacOS, Linux. Retrieved August 16, 2023 from https://support.zoom.us/hc/en-us/articles/201362023-Zoom-system-requirements-Windows-macOS-Linux

[47] William Vickrey. 1961. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance* 16, 1 (1961), 8–37.

[48] Arun Viswanathan, Eric C. Rosen, and Ross Callon. 2001. Multiprotocol Label Switching Architecture. RFC 3031. doi:10.17487/RFC3031

[49] Sam M. Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J. Knottenbelt. 2022. SoK: Decentralized Finance (DeFi). arXiv:2101.08778 [cs.CR]

[50] Wikipedia. 2024. Network Time Protocol. Retrieved 2 September 2024 from https://en.wikipedia.org/wiki/Network_Time_Protocol

[51] John T. Wroclawski. 1997. *The Use of RSVP with IETF Integrated Services.* RFC 2210. IETF. doi:10.17487/RFC2210
[52] Karl Wüst and Arthur Gervais. 2016. *Ethereum eclipse attacks.* Technical Report. ETH Zurich. doi:10.3929/ethz-a-010724205
[53] Marc Wyss, Giacomo Giuliari, Markus Legner, and Adrian Perrig. 2021. Secure and Scalable QoS for Critical Applications. In *IEEE/ACM International Symposium on Quality of Service (IWQoS).*
[54] Marc Wyss, Giacomo Giuliari, Jonas Mohler, and Adrian Perrig. 2022. Protecting Critical Inter-Domain Communication through Flyover Reservations. In *ACM Conference on Computer and Communications Security (CCS).* Association for Computing Machinery, New York, NY, USA, 2961–2974. doi:10.1145/3548606.3560582
[55] Karl Wüst and Arthur Gervais. 2018. Do you Need a Blockchain?. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT).* doi:10.1109/CVCBT.2018.00011
[56] Abraham Yaar, Adrian Perrig, and Dawn Song. 2004. SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy (S&P).*
[57] Xiaowei Yang, David Wetherall, and Thomas Anderson. 2005. A DoS-limiting network architecture. In *ACM SIGCOMM Conference.* doi:10.1145/1080091.1080120
[58] Sharada Yeluri. 2023. Sizing router buffers — small is the new big. Retrieved August 16, 2023 from https://blog.apnic.net/2023/03/06/sizing-router-buffers-small-is-the-new-big/

Appendices

Appendices are supporting material that has not been peer-reviewed.

A Specification for Hummingbird on SCION

In this appendix, we include the specification of a new SCION [9] path type that supports Hummingbird reservations. SCION is well suited to deploy a bandwidth-reservation system such as ours, since it provides path choice to the source, which guarantees the stability of the path. In contrast, path stability in the current Internet with BGP routing is only provided as long as there is convergence in the network. For example, path stability is not guaranteed in the presence of BGP hijacking attacks [4].

This header specification is written analogously to the SCION header specification[7] and duplicates/reuses some of the information. Changes compared to the SCION header specification are indicated. The header layout for this path type is shown in Fig. 6; the individual parts are described in further detail in the following subsections.

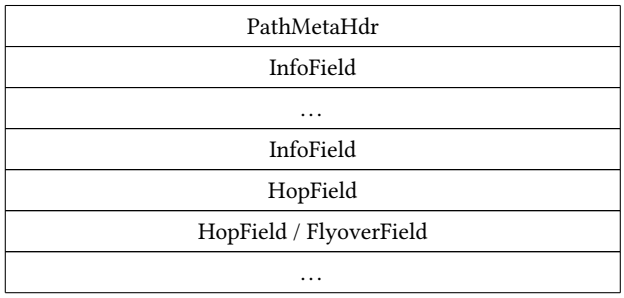

Figure 6: Path header layout for the Hummingbird path type consisting of a path meta header, up to 3 info fields, and up to 64 hop fields or flyover fields.

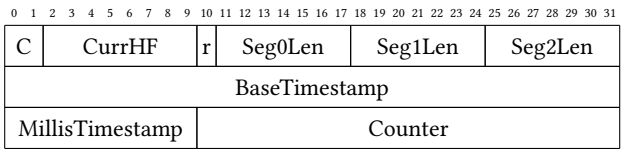

Figure 7: Layout of the PathMetaHdr.

A.1 Path Meta Header (changed)

The PathMetaHdr is a 12-byte header containing meta information about the SCION path contained in the path header, see Fig. 7. It contains the following fields:

(C)urrINF 2-bit index (0-based) pointing to the current info field (see offset calculations below).

CurrHF (changed) 8-bit index (0-based) pointing to the start of the current hop field (see offset calculations below) in 4-byte increments. This index is increased by 3 for normal hop fields and by 5 for flyover hop fields, which are 12 B and 20 B long, respectively.

r Unused and reserved for future use.

Seg{0,1,2}Len (changed) 7-bit encoding of the length of each segment. The value in these fields is the length of the respective segment in bytes divided by 4. $Seg_iLen > 0$ implies the existence of info field i.

BaseTimestamp (new) A unix timestamp (unsigned integer, 1-second granularity, similar to beacon timestamp in normal SCION path segments) that is used as a base to calculate start times for flyovers and the high granularity MillisTimestamp.

MillisTimestamp (new) Millisecond granularity timestamp, as offset from BaseTimestamp. Used to compute MACs for flyover hops and to check recentness of a packet.

Counter (new) A counter for each packet that is sent by the source to ensure that the tuple (BaseTimestamp, MillisTimestamp, Counter) is unique. This can then be used for the optional duplicate suppression at an AS.

Path Offset Calculations (changed). The number of info fields is implied by $Seg_iLen > 0, i \in [0,2]$, thus $NumINF = N + 1$ where $N = \max_{i\in[0,2]}$ s.t. $\forall j \leq i : Seg_jLen > 0$. It is an error to have $Seg_XLen > 0 \wedge Seg_YLen = 0$ for $X > Y$. If all $Seg_iLen = 0, i \in [0,2]$, then this denotes an empty path, which is only valid for intra-AS communication.

The offsets of the current info field and current hop field (relative to the end of the address header) are now calculated as

$$InfoFieldOffset = 12\,\text{B} + 8\,\text{B} \cdot CurrINF, \tag{5a}$$

$$HopFieldOffset = 12\,\text{B} + 8\,\text{B} \cdot NumINF + 4\,\text{B} \cdot CurrHF. \tag{5b}$$

To check that the current hop field is in the segment of the current info field, the CurrHF needs to be compared to the SegLen fields of the current and preceding info fields.

A.2 Info Field (unchanged)

The format of an InfoField is the same as in the SCION path type and shown in Fig. 8. It contains the following fields:

r Unused and reserved for future use.

[7] https://docs.scion.org/en/latest/protocols/scion-header.html

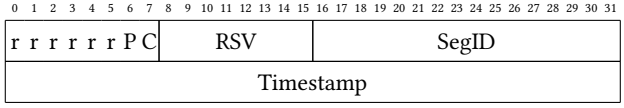

Figure 8: Layout of an InfoField.

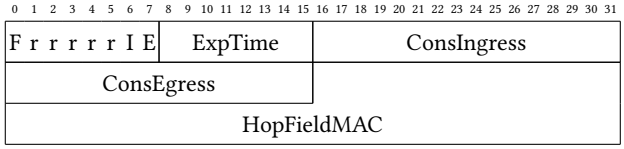

Figure 9: Layout of a HopField.

P Peering flag. If set to true, then the forwarding path is built as a peering path, which requires special processing on the data plane.

C Construction direction flag. If set to true then the hop fields are arranged in the direction they have been constructed during beaconing.

RSV Unused and reserved for future use.

SegID Updatable field used in the MAC-chaining mechanism.

Timestamp Timestamp created by the initiator of the corresponding beacon. The timestamp is expressed in Unix time, and is encoded as an unsigned integer within 4 bytes with 1-second time granularity. It enables validation of the hop field by verification of the expiration time and MAC.

A.3 HopField (slightly changed)

The HopField, used for hops without a reservation, is slightly changed compared to SCION and is shown in Fig. 9. This HopField is also used as the first hop field for reserved hops at segment boundaries (see Appendix A.5). It contains the following fields:

F (new) Flyover bit. Indicates whether this is a hop field or a flyover hop field. Set to 0 for HopFields.

r (unchanged) Unused and reserved for future use.

I (unchanged) ConsIngress Router Alert. If the ConsIngress Router Alert is set, the ingress router (in construction direction) will process the L4 payload in the packet.

E (unchanged) ConsEgress Router Alert. If the ConsEgress Router Alert is set, the egress router (in construction direction) will process the L4 payload in the packet.

ExpTime (unchanged) Expiry time of a hop field. The field is 1-byte long, thus there are 256 different values available to express an expiration time. The expiration time expressed by the value of this field is relative, and an absolute expiration time in seconds is computed in combination with the timestamp field (from the corresponding info field).

ConsIngress, ConsEgress (unchanged) The 16-bit interface IDs in construction direction.

HopFieldMAC (name changed) 6-byte MAC to authenticate the hop field. For details on how this MAC is calculated refer to the hop-field MAC computation of the SCION path type.[8]

[8] https://docs.scion.org/en/latest/protocols/scion-header.html#hop-field-mac-computation

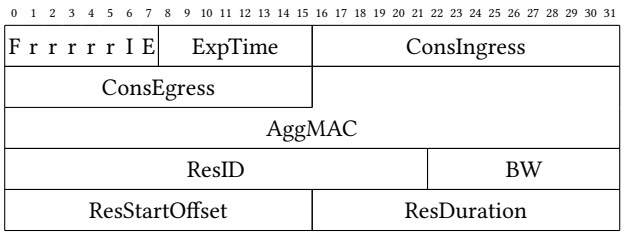

Figure 10: Layout of a FlyoverHopField.

A.4 FlyoverHopField (new)

The FlyoverHopField shown in Fig. 10 is present if the reservation bit in the previous hop field is set to 1. It contains the following fields:

F Flyover bit. Indicates whether this is a hop field or a flyover hop field. Set to 1 for FlyoverHopFields.

r, I, E, ExpTime, ConsIngress, ConsEgress These values are the same as in the standard HopField. Note that ExpTime is the expiration time of the standard HopField, not the expiration time of the reservation.

AggMAC The aggregate MAC [26] (i.e., XOR) of the standard HopField MAC and the per-packet flyover MAC as described in Section 4:

$$AggMAC = HopFieldMAC \oplus FlyoverMAC, \tag{6}$$

where

$$FlyoverMAC = \mathsf{PRF}_{A_K}(DstAddr \parallel PktLen \parallel TS)\ [:6], \tag{7a}$$

$$TS = ResStartOffset \parallel MillisTimestamp \parallel Counter, \tag{7b}$$

$$DstAddr = DstISD \parallel DstAS, \tag{7c}$$

$$PktLen = PayloadLen + 4 \cdot HdrLen. \tag{7d}$$

PayloadLen and HdrLen are the values from the SCION Common Header and PktLen is a 2-byte value. If an overflow occurs during the calculation of PktLen, the packet must be dropped. A_K is computed as described in Eq. (2) and Appendix A.6. The bit-layout for the input to the MAC computation (Eq. (7a)) is shown in Fig. 11.

ResID 22-bit Reservation ID, this allows for approximately 4 million concurrent reservations for a given ingress/egress pair.

BW 10-bit bandwidth field indicating the reserved bandwidth which allows for 1024 different values. The values could be encoded similarly to floating point numbers (but without negative numbers or fractions), where some bits encode the exponent and some the significant digits. For example, one can use 5 bits for the exponent and 5 bits for the significand and calculate the value as significand if `exponent = 0` or otherwise as `(32 + significand) << (exponent - 1)`. This allows values from 0 to almost 2^{36} with an even spacing for each interval between powers of 2.

ResStartOffset The offset between the BaseTimestamp in the Path Meta header and the start of the reservation (in seconds). This allows values up to approximately 18 hours in second granularity.

ResDuration Duration of the reservation, i.e., the difference between the timestamps of the start and expiration time of the reservation.

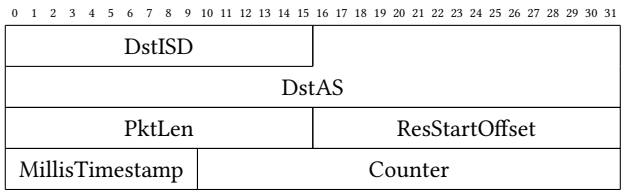

Figure 11: Layout of the input to the computation for the FlyoverMAC (see Eq. (7a)).

Algorithm 2 Authenticating and forwarding traffic at the ingress border router of the ith on-path AS (AS i). For a higher-level overview, see Fig. 13. Fields inside the packet (*pkt*) are represented as $\boxed{Field}$. SV_i is the AS-local secret value for the computation of Hummingbird authentication tags.

Input: pkt, SV_i, K_i
1: **if** flyover bit $\boxed{\text{F}}$ is set **then**
2: (retFlyover, TS, PktLen) ← FlyoverProcessing(pkt, SV_i) ▷ Algorithm 3
3: **else**
4: retFlyover ← `fwd_best_effort`
5: **if** retFlyover == `drop_pkt` **then**
6: **drop packet**
7: retHf ← StandardHfProcessing(pkt, K_i) ▷ Algorithm 4
8: **if** retHf == `drop_pkt` **then**
9: **drop packet**
10: **if** retFlyover == `fwd_flyover` **then**
11: retMonitor ← BandwidthMonitoring($\boxed{ResID}$, $\boxed{BW}$, *PktLen*) ▷ Algorithm 4
12: **if** retMonitor == `fwd_flyover` **then**
13: **forward packet as high priority**
14: **forward packet as best effort**

A.5 Segment boundaries

On segment boundaries, the AS needs to process two hop fields (last HF in the first segment, first HF in the second segment). Both HFs together specify the ingress and egress interfaces. If there is a flyover field for this AS, it must be placed in the first segment as the first HF of the AS.

A.6 Key Derivation

Since the derivation of A_K is done completely by AS K, they are free to choose the key derivation function used in the key derivation as described in Eq. (2) in the main paper:

$$A_K = \mathsf{PRF}_{SV_K}\left(ResInfo_K\right),$$

The concrete input to the key derivation is shown in Fig. 12, where $ResStart = BaseTimestamp - ResStartOffset$.

A.7 Packet Forwarding

When forwarding a packet with a FlyoverHopField at the current AS, each ingress border router executes the steps described in Section 4.3 in addition to the packet processing of the standard SCION

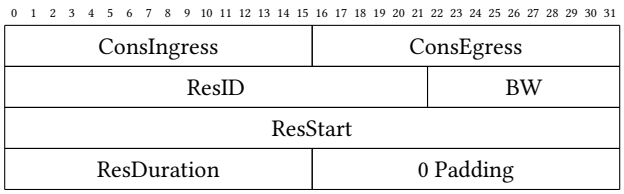

Figure 12: Layout of $ResInfo_K$ used for the key derivation in Eq. (2).

Algorithm 3 Reservation processing at a border router. Δ is the maximum packet age, and δ is the maximum acceptable clock skew.

Input: pkt, SV_i
Returns: (fwd_type, *TS*, *PktLen*)
function FlyoverProcessing(pkt, SV_i)
$ResStart \leftarrow \boxed{BaseTimestamp} - \boxed{ResStartOffset}$
$ResInfo_i \leftarrow \boxed{ConsIngress} \,\|\, \boxed{ConsEgress} \,\|\, \boxed{ResID} \,\|\, \boxed{BW} \,\|\, ResStart \,\|\, \boxed{ResDuration}$ ▷ Fig. 12
$A_i \leftarrow \mathsf{PRF}_{SV_i}(ResInfo_i)$ ▷ Eq. (2)
$TS \leftarrow \boxed{ResStartOffset} \,\|\, \boxed{MillisTimestamp} \,\|\, \boxed{Counter}$ ▷ Eq. (7b)
$DstAddr \leftarrow \boxed{DstISD} \,\|\, \boxed{DstAS}$ ▷ Eq. (7c)
(overflow, *PktLen*) ← checkedAdd($\boxed{PayloadLen}$, $4 \cdot \boxed{HdrLen}$) ▷ Eq. (7d)
if overflow **then**
Return: (`drop_pkt`, *TS*, *PktLen*)
$FlyoverMAC_i \leftarrow \mathsf{PRF}_{A_i}(DstAddr \,\|\, PktLen \,\|\, \text{TS})[:6]$ ▷ Eq. (7a)
$\boxed{AggMAC_i} \leftarrow FlyoverMAC_i \oplus \boxed{AggMAC_i}$ ▷ Cand. HF MAC
$absTS \leftarrow \boxed{BaseTimestamp} \,\|\, \boxed{MillisTimestamp}$
if $now() - absTS \notin [-\delta, \Delta + \delta]$ **then** ▷ Freshness check
Return: (`fwd_best_effort`, *TS*, *PktLen*)
$ResExp \leftarrow ResStart + \boxed{ResDuration}$
if now() $\notin$ [*ResStart*, *ResExp*] **then** ▷ Res. active check
Return: (`fwd_best_effort`, *TS*, *PktLen*)
Return: (`fwd_flyover`, *TS*, *PktLen*)

path type with the difference that the HopFieldMAC and the FlyoverMAC are XORed before comparing them to the value stored in the header. The FlyoverMAC is computed as shown in Eq. (7a).

Before forwarding, the router then replaces the AggMAC in the Flyover with the HopFieldMAC that it computes during the verification of the FlyoverHopField. This allows the path to be reversed easily.

A high-level overview is shown in Fig. 13, the detailed algorithms are specified in Algorithms 2 to 4.

In the *Reservation active check* in Algorithm 3, the clock skew is not included, since this could cause problems at the boundaries of two reservations that share the same *ResID*. In particular, it may i) cause packets from expired reservations to be counted towards a new, unrelated reservation, or ii) if no adjacent reservations are

Algorithm 4 Overview of the standard SCION border router processing steps. Providing a full description of SCION border router processing is beyond the scope of this document. This algorithm highlights the border router checks that are essential to Hummingbird's security properties. K_i is the AS-local secret value for the computation of the SCION HopFieldMACs.

Input: *pkt*, K_i
Returns: fwd_type

function StandardHfProcessing(*pkt*, K_i)
  if hop field is expired **then**
    Return: `drop_pkt`
  Recompute *HopFieldMAC* using *pkt*, K_i
  if *HopFieldMAC* in *pkt* does not match **then**
    Return: `drop_pkt`
  Additional checks (segment combination allowed, ...)
  Update the $\boxed{SegID}$
  if flyover bit $\boxed{F}$ is set **then**
    $\boxed{CurrHF} \leftarrow \boxed{CurrHF} + 5$
  else
    $\boxed{CurrHF} \leftarrow \boxed{CurrHF} + 3$
  Return: **fwd**

assigned the same *ResID*, more traffic may be prioritized than the AS has capabilities for.

A.8 Path Reversal

Path reversal works in the same way as standard SCION with the addition that for each FlyoverHopField, the flyover bit is set to 0, all fields that are unique to FlyoverHopFields are removed (ResID, BW, ResStartOffset, ResDuration) to convert the FlyoverHopField to a regular HopField, and the Seg_iLen values are adjusted accordingly. This provides a valid path of the Hummingbird path type (albeit without reservations), but it can further be converted to the regular SCION path type by replacing the PathMetaHdr with the PathMetaHdr of the regular SCION path type (i.e., removing the timestamps and and converting the Seg_iLen values).

B Additional Evaluation Results

B.1 Gas Cost Evaluation

Table 2 (of which the results table in Table 1 is a subset) shows the cost of the contract calls for our control plane. Since the cost is deterministic, the table does not include any measurement uncertainties. The cost for transactions in Sui is split into three components: Computation cost, storage cost, and a storage rebate. Computation cost is based on the complexity of the computation, which is charged according to fixed buckets of computational units. These computational units are then converted to a value in Sui according to the *computation gas price* provided in the transaction. In our results, the values are provided based on the current reference gas price on mainnet of 7.5×10^{-7} SUI per unit. Storage of objects is charged based on a *storage gas price*, which is currently 7.6×10^{-6} SUI per byte. The third component is a storage rebate, which the transaction sender receives when deleting items from the storage, and amounts to 99% of the original value paid in SUI for the storage of the item. If the storage rebate exceeds the transaction's computation and storage cost, the sender receives the difference.

As Table 2 shows, the cost of transactions for individual calls for interacting with the assets or the marketplace are relatively cheap, costing only fractions of a cent. The cost of buying a full path depends on the length of the path, and it is dominated by the cost of buying multiple assets, since this incurs the storage cost of splitting an asset and re-listing the pieces that are not bought. In our benchmark, each asset on a path requires a worst-case split, i.e., the buyer buys a time interval from the middle of the time interval represented by the listing and only a fraction of the bandwidth. This causes two splits in the time dimension and one in the bandwidth dimension.

Since buying an asset creates new objects and delivering a reservation deletes them, most of the storage fee incurred by the buyer is later refunded *to the AS* as a storage rebate. To lower the transaction cost for the buyer, the market could be pre-loaded with funds provided by the AS, which then the smart contract could use to refund part of the cost of buying assets. The AS later gets these funds back through the storage rebate when deleting the assets.

B.2 Border Router Processing

Table 3 reports the execution times of all the steps required to process a regular SCION packet, and the additional overhead incurred when processing a Hummingbird packet (darker gray background). A SCION packet requires 123 ns to be processed, while Hummingbird adds an overhead of 185 ns for a total of 308 ns.

B.3 Traffic Generation Performance

To complete our data-plane benchmarks, we study the performance of a Hummingbird *traffic generator*. In most cases, we expect the sources to be end-hosts, where the forwarding performance is limited by the hosts' stacks and network uplinks. However, performant traffic generation is essential in scenarios where a single entity purchases bandwidth for hosts within its network—for example, a corporate LAN—and provides reservations to hosts through a dedicated Hummingbird gateway.

Figure 14 shows the traffic-generation throughput of a multi-core Hummingbird gateway implemented in DPDK. Increasing the number of cores enhances the throughput: For payloads of 500 B, a mere 32 cores deliver 160 Gbps line rate for both Hummingbird and SCION, even across long paths with eight on-path ASes. Table 4 details the durations of individual per-packet computation steps for a four-hop path, where the total overheads are 494 ns and 293 ns for Hummingbird and SCION, respectively.

Traffic generation at the source is notably slower than traffic forwarding at the border router. This is as expected: The source has to compute the Hummingbird packet authentication tags for all ASes on the path, while on-path border routers only need to compute the authentication tag for their own AS. Nevertheless, a 32-core source gateway is sufficient to serve the vast majority of deployments, as residential and corporate link speeds seldom exceed 1 Gbps, and even ISP links are usually below 100 Gbps.

Figure 15 demonstrates the single-core packet generation performance at the source. For both SCION and Hummingbird traffic, the throughput achieved correlates with the number of ASes on the path

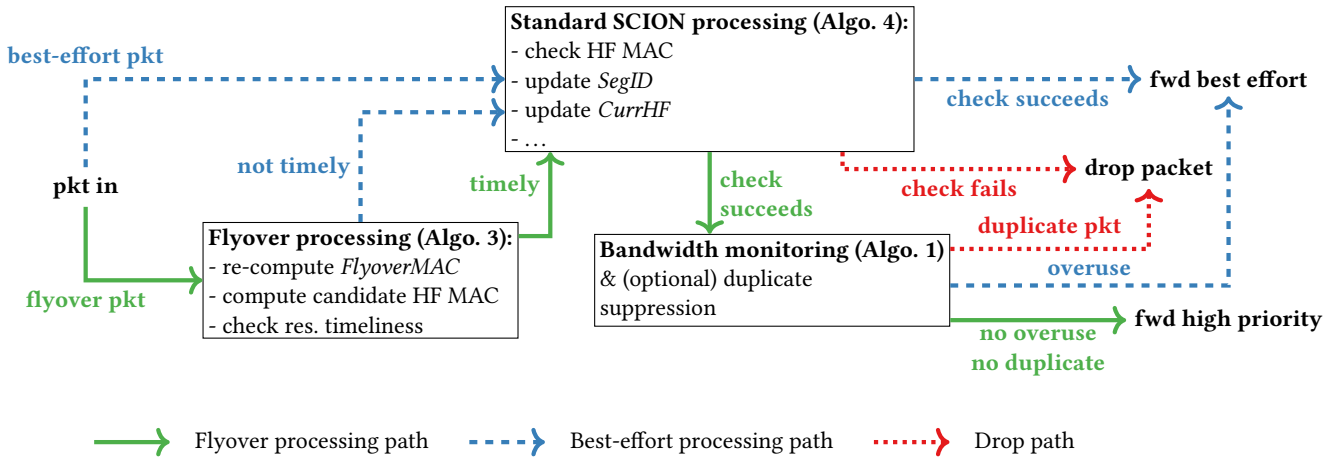

Figure 13: High-level representation of the packet-processing pipeline at the border router.

Table 2: Gas cost (rounded to two significant figures) for contract calls to asset and marketplace contracts, as well as transactions for atomically buying and redeeming a full path. A Negative value indicates that the caller *earns* SUI because the storage rebate exceeds the transaction cost.

Contract call	Computation (SUI)	Storage (SUI)	Storage rebate (SUI)	Total (SUI)	Total (USD)*
Asset functions					
issue	0.000 75	0.0046	0.0025	0.0029	0.0035
split_time	0.000 75	0.0052	0.0031	0.0029	0.0035
split_bandwidth	0.000 75	0.0052	0.0031	0.0029	0.0035
fuse_time	0.000 75	0.0031	0.0051	−0.0013	−0.0016
fuse_bandwidth	0.000 75	0.0031	0.0051	−0.0013	−0.0016
redeem	0.000 75	0.0045	0.0051	0.000 12	0.000 14
deliver_reservation	0.000 75	0.000 99	0.0045	−0.0027	−0.0033
Market functions					
create_marketplace	0.000 75	0.0030	0.000 98	0.0028	0.0034
register_seller	0.000 75	0.0026	0.000 98	0.0024	0.0029
create_listing	0.000 75	0.011	0.0067	0.0050	0.0061
buy (full)	0.000 75	0.0071	0.010	−0.0023	−0.0028
buy (split bw)	0.000 75	0.013	0.010	0.0039	0.0048
buy (split time)	0.000 75	0.020	0.010	0.010	0.012
buy (split both)	0.000 75	0.026	0.010	0.016	0.020
Atomic buy-and-redeem					
1 hop	0.000 75	0.047	0.016	0.031	0.038
2 hops	0.000 75	0.090	0.029	0.062	0.076
4 hops	0.000 75	0.18	0.054	0.12	0.15
8 hops	0.0015	0.35	0.10	0.25	0.30
16 hops	0.0030	0.69	0.20	0.49	0.60

Computation price: 7.5×10^{-7} SUI/unit; storage price: 7.6×10^{-6} SUI/byte;
SUI price: 1.221 USD (as of 2024-04-18 14:09 UTC);

and scales proportionally to the packets' payload size. For instance, with a 1 kB payload size and flyovers on four on-path ASes, Hummingbird achieves a throughput of 17.90 Gbps, whereas SCION's best-effort traffic reaches 28.64 Gbps. However, with smaller 100 B payloads, the throughput decreases to 4.65 Gbps and 7.70 Gbps, respectively.

C Bidirectional Reservation Support

Bidirectional reservations can generally be implemented by having the source communicate pre-computed hop authentication tags to the destination with every packet. If the reverse traffic is expected to be larger, multiple tags per hop can be provided in each packet.

Table 3: Fine-grained packet validation and forwarding timings at the border router. The timings are independent of the number of on-path ASes and the size of the payload.

Task	Time [ns]
Check packet size	14
Parse packet headers	30
Check whether hop field is expired	8
Recompute SCION hop field MAC	46
Update segment identifier (SegID)	4
Update current hop field pointer	13
Check if hop field is of type SCION or Flyover	8
Compute absolute start of reservation (ResStart)	8
Compute authentication key (A_i)	43
AES-extend authentication key (A_i)	24
Validate high-precision time stamp	6
Recompute flyover MAC	44
Compute aggregate MAC	4
Verify xor-ed MAC same as in header	9
Check whether the reservation is still active	8
Check for overuse	39
Total	**123 [+185]**

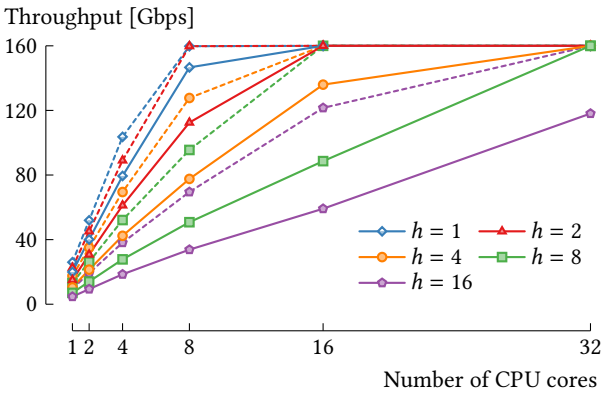

Figure 14: Packet generation performance at the source for packets carrying a payload of 500 B, and for different number of AS-level hops (h) and cores. Solid lines correspond to Hummingbird reservations, dashed lines to standard SCION best-effort traffic.

However, this approach has two main problems: (i) sending additional authentication tags for each hop on the path in almost every packet introduces a large overhead, and (ii) this approach increases the complexity of monitoring, since it requires aggregating the monitoring information for forwards and backwards traffic, which are monitored at two different border routers.

We believe that bidirectional reservations are better obtained with a separate—and possibly out-of-band—exchange protocol:

- The source obtains reservations to the destination normally and obtains separate reservations for the reverse path.

Table 4: Fine-grained packet generation timings at the source, for four AS-level hops; the additional operations required for Hummingbird are highlighted. All timings are independent of the payload size except in the case of "Add packet payload", which we evaluate for 500 B and 1500 B payloads.

Task	Time [ns]	
Add Ethernet, IP, Scion header fields	107	
Compute flyover MACs (4 on-path ASes)	201	
Add hop fields for all on-path ASes	171	
Add 500 B (1500 B) payload	15	(40)
Total for 500 B (1500 B) payload	**494**	**(519)**

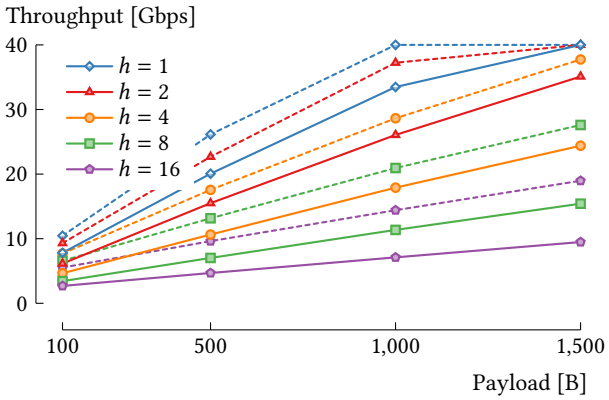

Figure 15: Single-core packet generation performance at the source for different number of AS-level hops (h) and payloads. Solid lines correspond to Hummingbird reservations, dashed lines to standard SCION best-effort traffic.

- The source communicates the authentication keys to the destination for the reverse path (if the destination is enabled to use the reservations).
- Source and destination use the reservations as normal.

Note that, even though on the data plane both source and destination use unidirectional "forward" reservations, these are both billed to the source and therefore act as a backward reservation. This is an important property enabled by the control-plane independence of Hummingbird: The source can obtain a reservation in any direction and for any hop, which is not the case in previous bandwidth-reservation protocols such as Colibri [17] or Helia [54].

With our control plane, the source could even send the bandwidth assets for the backward reservation to the destination (assuming it knows their on-chain identity), which would allow the destination to directly obtain the reservation information and authentication keys by automatically redeeming the assets. In this case, it may be useful to use additional tags in the asset that contain the source address to ensure that the (trusted) destination uses the reservations for communicating with the source.