Consent Routing: Towards Bilaterally Trusted Communication Paths

Mathias Blarer
ETH Zürich
blarerm@student.ethz.ch

Jonghoon Kwon
ETH Zürich
jong.kwon@inf.ethz.ch

Vincent Graf
ICRC
vgraf@icrc.org

Adrian Perrig
ETH Zürich
adrian.perrig@inf.ethz.ch

***Abstract*—In today's Internet, the security of data transfers largely depends on the forwarding path: on-path adversaries can launch powerful attacks against the confidentiality, integrity, and availability of Internet communication. Moreover, current routing protocols give little path control to end hosts; at best, a multi-homed host can choose the first hop of the forwarding path. In short, communicating hosts are facing the problem that they need to trust the entities which forward their packets but can barely choose the forwarding path. Recent research in networking has shown that path-aware network architectures can give the sender control over the path selection while increasing the overall efficiency and security of the network. Still, only half of the trust problem is solved: in these architectures, path selection is up to the sender's judgment, even though the sender *and* the receiver have the same vital interest in choosing the forwarding path for their communication. In this paper, we introduce consent routing, a new routing paradigm in which the consent of both the sender and the receiver is required prior to using a forwarding path. The novelty of consent routing is to make path selection a cooperative process between the distributed communicating parties, enabling new opportunities for security and trust, e.g., mitigation of surveillance, censorship, and traffic analysis. Our implementation shows that consent routing is feasible in practice and can be incrementally deployed without changes to the underlying network architecture.**

I. Introduction

The reliability of heterogeneous distributed systems is challenged by the dynamic routing of today's Internet, e.g., the IP prefix hijacking in Border Gateway Protocol (BGP) allows an autonomous system (AS) to illegitimately reroute and then intercept traffic [7], [10]. In recent years, many new, so-called path-aware network (PAN) architectures have been proposed [8], [17], [30], [38], [42] as an improvement over today's BGP-based Internet, allowing the sender of a packet to choose among a set of forwarding paths. In such a network, the packet source is able to forward the packet on the "safest", most trusted path, or more generally on the path that best suits the sender's purpose.

Yet, our main observation is that this is not sufficient. Ideally, a forwarding path should suit the sender's *and* the receiver's purpose. If sender and receiver have the same needs and trust the same network paths, then every sender-chosen path will be appropriate. In the opposite case, a forwarding path chosen by the sender could raise security concerns on the receiver side if it traverses countries or nodes that are not trusted by the receiver. The goal of routing should therefore be to forward packets on a path that is trusted and accepted by both communicating hosts, if such a path exists.

Consent routing is the instantiation of this idea: before using a given forwarding path, both sender and receiver must provide their respective consent. By introducing this notion of consent routing, we are the first to raise and address the need to base routing decisions on the trust assumptions, and more generally on the needs of both communicating hosts.

Our ConPASS protocol allows two hosts s and t to achieve consent routing in a path-aware network by exchanging selected forwarding paths and their respective consent for using those paths. In simplified terms, the protocol works as follows: the protocol initiator (s) takes all available forwarding paths between s and t, and sends to the protocol responder (usually t) those paths that s consents to use; t also filters the received paths with its own consent logic and returns the resulting paths to s. After the protocol run, s and t both know the common set of accepted forwarding paths; and given that s and t are in a path-aware network, they can from now on select such a path for sending packets between them.

The main challenges for designing a consent routing mechanism are (i) to keep the overhead prior to and during communication low and, at the same time, (ii) to preserve the confidentiality of every host's consent logic, which may contain sensitive information such as the network nodes or countries to avoid. In the ConPASS protocol, both challenges are addressed simultaneously. On the one hand, the trust assumptions and constraints that make up a host's consent logic are never explicitly shared. On the other hand, routing overhead is significantly reduced by not giving consent to all possible end-to-end paths but to path *segments*, shorter and combinable building blocks of end-to-end paths.

We provide an implementation of ConPASS and evaluate its performance to demonstrate the practical viability and scalability of consent routing. The measurements within a realistic network topology show that the average latency overhead of running the ConPASS protocol—needed *once* when connecting with a host for the first time—amounts to 20–180 ms (given an RTT $\leq$ 25 ms), where only 20–65 ms are actual processing overhead, the remaining delay of up to 115 ms comes from the additional TCP and TLS handshakes. The traffic overhead generated by a protocol run is also comparatively low, for a realistic network size less than $1/24$ of the amount of data that is downloaded from a medium-sized website on first contact.

This paper makes the following contributions:

- We address the necessity of making routing decisions that suit the needs of both communicating peers by requiring their respective consent before using a path. We present this concept under the name *consent routing*.

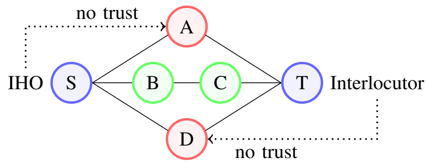

Figure 1. Sample network topology between an international humanitarian organization and an interlocutor, exhibiting heterogeneous trust assumptions.

- We propose CONPASS, an application-layer protocol that allows two path-aware hosts to exchange consent for forwarding paths, as a possible consent routing approach.
- We provide an open-source implementation of CONPASS running on top of an existing path-aware architecture and show that consent routing is feasible in practice.

II. PROBLEM DEFINITION

Through a case study, we explore the dynamics of heterogeneous trust assumptions in long-distance Internet connections. We summarize the desired properties of such connections in a concept called consent routing and derive the main requirements for routing approaches that implement this concept.

A. Case Study

Consider an international humanitarian organization (IHO) that seeks the confidential bilateral dialogue with parties in armed conflicts and other situations of violence [26]. When face-to-face meetings are not possible, Internet-based voice or video calls between IHO staff and representatives of the interlocutor offer a viable alternative as long as both parties trust that the communication channel does not leak confidential information to an adversarial third party and is not subject to censorship attempts inside the network.

Figure 1 shows the network topology between the IHO's AS-level network S and the interlocutor's network T in a fictive communication scenario. Network packets can be sent over one of three possible paths: S-A-T, S-B-C-T, or S-D-T. In our scenario, the ASes are generally trusted by both parties, with two exceptions: since the IHO does not enjoy legal immunity in AS A's country, it tries to avoid the path through A. AS D, on the other hand, is known to cooperate with one of the interlocutor's enemies, hence the interlocutor does not trust D nor any path on which D lies. As a consequence, the only network path that is trusted by both entities is S-B-C-T.

Current inter-domain routing protocols are in general unable to select the forwarding path S-B-C-T that is fully trusted by both communication endpoints. In absence of particular incentives, BGP selects one of the shorter paths S-A-T and S-D-T, violating either the IHO's or the interlocutor's trust model. Since both paths are equally likely, neither the IHO nor the interlocutor can have trust in the communication between them. A path-aware networking approach as in SCION [42] improves over BGP in that the source's trust model can steer the path selection. Whenever source and destination have the same trust assumptions, unilaterally trusted paths are also bilaterally trusted. Yet in our example, the trust assumptions are heterogeneous. Assuming that each party sends out packets on the shortest trusted path, the IHO would send packets over S-D-T and the interlocutor over S-A-T, leading to a situation that violates each receiver's path requirements.

If as a consequence the IHO and its interlocutor fail in establishing a confidential communication channel over a trusted Internet path, it is not due to the absence of such paths but to the lack of support from current routing schemes.

B. Consent Routing

The case study illustrates the need for more sophisticated routing approaches that take into account the trust assumptions (and possibly other constraints) of both the source and destination such that a bilaterally trusted and accepted network path is used for communication, if such a path exists.

At its core, routing is about the management of forwarding paths between individual hosts in a network. It is a decision-making process to determine the sequence of network nodes (i.e., routers) through which data packets will pass from one host to another. An important aspect of routing mechanisms is to define who can be part of this process and how the decision is reached. In this paper, we introduce the term *consent routing* as the notion for routing approaches where both the source and the destination are involved in the routing process and can consent to the usage of particular forwarding paths. In our definition of consent routing, we consider unidirectional paths, which is a more general concept than bidirectional paths.

Definition. To define consent routing more formally, we let P be the available set of paths from a source s to a destination t, and let $P_{s\to}$ and $P_{\to t}$ be the subsets of P that respectively have the consent of s and t:

$$P_{s\to} := \{p \in P \mid s \text{ consents to send over } p\},$$
$$P_{\to t} := \{p \in P \mid t \text{ consents to receive over } p\}.$$

Furthermore, let $P_{s\to t}$ be the paths that are in $P_{s\to}$ as well as in $P_{\to t}$ and thus have the consent of both s and t:

$$P_{s\to t} := P_{s\to} \cap P_{\to t}.$$

A routing scheme forwarding traffic from s to t over a network path p is a consent routing scheme if $P_{s\to t} \neq \emptyset$ implies that $p \in P_{s\to t}$. We then call p a *consent-routed* path.

C. Requirements

Apart from the functional requirements given by the definition above, a consent routing mechanism should also address the following non-functional requirements:

Security and Reliability. The routing mechanism needs to work reliably. In particular, the routing should not be susceptible to adversarial manipulation, and an off-path attacker should be unable to disrupt the routing process.

Confidentiality of Consent Logic. The logic according to which a host grants or denies consent to forwarding paths may contain sensitive information, including the host's trust assumptions. The routing system should therefore minimize the amount of such information that is intentionally shared or unintentionally leaked.

Low Communication Overhead. Finding a consent-routed path should require minimal latency and traffic overhead.

Scalability. The system should scale with the network size, both in terms of the number and length of network paths.

Incremental Deployment. Deploying the routing mechanism should require little changes to existing network infrastructure and yield added value for early adopters.

III. Background and Definitions

CONPASS can be used to achieve consent routing on top of path-aware network architectures, which are a type of source routing system. It can also leverage the fact that some path-aware architectures construct end-to-end paths from smaller segments, to be more performant in those architectures.

Source routing is a routing technique in which the sender of a packet determines how the packet is forwarded to the destination; the sender explicitly lists a sequence of nodes or processing instructions in the packet header such that the router at each hop forwards the packet according to the next node's address or by executing the given instruction. It has been used in various routing schemes [13], [15], [16], [21], [34], including IPv4 with the Loose and Strict Source Routing options [3] and IPv6 with the Routing Header extension [12].

Path-aware networking architectures are a special case of source routing architectures, suitable for inter-domain routing. In source routing, there is no general rule on how the sender determines the forwarding path. In a PAN architecture, network endpoints cannot create arbitrary paths; they need to choose between AS-level network paths made available to them through their network providers. This ensures that the chosen network paths comply with the routing policies of the intermediate ASes. Furthermore, the notion of packet-carried forwarding state instructs the data plane to forward packets along the chosen network path. Various PAN architectures have been developed in recent years, including Pathlet Routing [17], NIRA [38], SCION [42], and Route Bazaar [8].

Path segments are fragments of network paths that sources in PAN architectures concatenate into end-to-end paths. Intuitively, path segments are flexible and policy-compliant building blocks that network providers make available to hosts and from which a large number of paths can be constructed. The concept of path segments appears in many path-aware architectures, sometimes under other names. In NIRA, paths are built from two segment types, so-called uphill-segments and downhill-segments [38], such that the number of end-to-end paths is quadratic in the number of segments. Similarly, in SCION, there are up-segments and down-segments, likely connected through a third segment type, core-segments [29]. The number of possible paths is therefore at most cubic in the number of segments. In Pathlet Routing, segments are called pathlets and paths can be freely constructed from any number of segments, leading to a potentially exponential number of paths [17]. Path segments reduce redundant path information while providing a large number of paths to end hosts.

IV. The CONPASS Protocol

This section describes the design of CONPASS (**Co**nsent **N**egotiation over **Pa**th **S**egment**s**), a simple yet extensible protocol that allows two parties to negotiate a set of path segments that have the consent of both parties.

A. Overview

Objectives. The principal goal of CONPASS is to allow a source host s and a destination host t to negotiate a set of path segments on which both s and t consent to respectively send and receive packets. After the protocol run, the source s knows which of the exchanged path segments are accepted by the destination t and vice versa.

A secondary goal of CONPASS is to allow extending the protocol with new functionality, including the transport of segment metadata, delegating the consent exchange to other hosts, or cryptographically signing the consent per-segment as a proof-of-consent.

Participants. The CONPASS protocol follows a simple request–response scheme with two participants: the initiator and the responder. In most cases, the initiator role corresponds to the source host and the responder role to the destination host; the responder role is, however, more flexible and can be delegated to a different host (see Sections IV-C and VII).

Communication Flow. CONPASS achieves consent routing for two hosts s and t in a PAN architecture as follows:

1) The source s knows a set Σ of path segments available towards the destination t and establishes a secure and reliable channel with t, e.g., over TCP/QUIC with TLS.
2) As the protocol initiator, s sends t the set $\Sigma_{s\to} \subseteq \Sigma$ of segments to which s consents, over the secure channel.
3) As the protocol responder, t receives $\Sigma_{s\to}$ and responds with $\Sigma_{s\to t}$, the segment subset of $\Sigma_{s\to}$ to which t also consents, over the same secure connection.
4) s receives the set of segments $\Sigma_{s\to t}$ and uses them to construct all possible end-to-end paths $P_{s\to t}$ from s to t that have the consent of both s and t. Now s can use one of these consent-routed paths $p \in P_{s\to t}$ to reach t.

It is important to note that the communication between s and t uses consent-routed paths only *after* the protocol run. The paths used for the consent negotiation are chosen unilaterally.

B. Detailed Design

CONPASS is a generic protocol that can be used on top of any PAN architecture. Consequently, parts of the design need to be defined and implemented specifically for the underlying network architecture. For the sake of generality, the protocol design presented in this section remains on a generic level.

Protocol Messages. There are two types of protocol messages, REQUEST and RESPONSE, sent by the initiator and responder, respectively. They have a similar structure:

$$\text{REQUEST} := (s \parallel t \parallel \sigma_1 \parallel \cdots \parallel \sigma_m), \tag{1}$$

$$\text{RESPONSE} := (\sigma_{m+1} \parallel \cdots \parallel \sigma_{m+n}), \tag{2}$$

where s and t are the source and destination nodes of the desired communication, and σ denotes a path segment; note

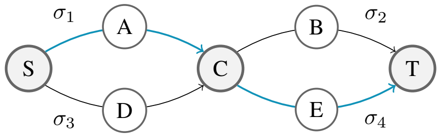

Figure 2. Network topology containing four path segments: $\sigma_1, \sigma_2, \sigma_3, \sigma_4$. A fifth segment, σ_5, is composed of σ_1 and σ_4.

that less important fields like the protocol version or the payload length are omitted for brevity. The representation of s, t, and σ depends on the network layer; for example, in the case of inter-domain routing, s and t are the respective identifiers of the source and destination ASes, and the path segments represent sequences of AS-level hops.

A path segment σ can be defined in one of two ways: as a *literal*, listing the sequence of network hops that constitute the segment, or as a *composition*, listing adjacent segments that are concatenated into a larger segment. Both segment types are illustrated in Figure 2; segments σ_1 through σ_4 are best described as segment literals (e.g., σ_1 := S-A-C) while σ_5 can be described simply as a composition of σ_1 and σ_4. The main idea of segment compositions is thus to allow the efficient re-use of segment literals. More formally,

$$\sigma := (\textsc{Type} \parallel \textsc{Consent} \parallel \textsc{Val}_1 \parallel \cdots \parallel \textsc{Val}_\ell), \tag{3}$$

$$\textsc{Val}_i := \begin{cases} \textsc{Hop}_i & \text{if } \textsc{Type} = \text{“literal”}, \\ \textsc{SegIdx}_i & \text{if } \textsc{Type} = \text{“composition”}. \end{cases} \tag{4}$$

Each segment is a list of values $\textsc{Val}_i$. In a segment literal, $\textsc{Val}_i$ denotes the i-th network hop $\textsc{Hop}_i$, e.g., $\textsc{Val}_2 = \text{A}$ in the segment literal σ_1 from Figure 2. As mentioned, the format of $\textsc{Hop}_i$ varies depending on the network layer. In a segment composition, $\textsc{Val}_i$ denotes the index of the i-th subsegment $\textsc{SegIdx}_i$, in the order of appearance in the protocol messages. Assuming that σ_1 through σ_4 are already specified in this order, we would have $\textsc{Val}_2 = 4$ in the segment composition σ_5 from Figure 2 since σ_5's second subsegment is σ_4. To avoid cyclic dependencies, a segment composition can only contain subsegments that appeared earlier in the protocol messages. In other terms, for any $\textsc{SegIdx}$ contained in a segment composition σ_k, it must hold that $\textsc{SegIdx} \in \{1, \ldots, k-1\}$. In particular, a segment composition in the RESPONSE can refer to a subsegment that appeared in the REQUEST.

Every path segment contains a boolean flag CONSENT; if CONSENT is “accept” (true), the segment has the message creator's consent. Otherwise, if the value is “deny” (false), the segment is not accepted on its own but is—directly or transitively—part of a larger segment that is accepted. Thus, hosts can allow only certain combinations of path segments.

In principle, all path segment can be expressed as a literal, but in many cases, this is inefficient. If the responder wants to acknowledge its consent of a path segment sent by the initiator, the responder does not need to include the whole path segment as a literal in the response. Instead, the responder can include a segment composition with a single segment index referring to the accepted path segment. Also, if multiple path segments contain the same subsequence of network hops, it can be more efficient to extract this sequence to a separate segment literal than to repeat the same sequence in multiple literals.

Algorithm 1: Consent negotiation from the initiator's perspective over initial path segments $\hat{\sigma}_1, \ldots, \hat{\sigma}_k$ between s and t. ϕ is the initiator's path segment filter.

```
Data: φ, s, t, σ̂_1, ..., σ̂_k
Result: σ̃_1, ..., σ̃_ℓ
begin
    σ_1, ..., σ_m ← filter σ̂_1, ..., σ̂_k with φ
    send REQUEST to responder according to Eq. 1
    get RESPONSE from responder according to Eq. 2
    σ̃_1, ..., σ̃_ℓ ← filter σ_{m+1}, ..., σ_{m+n} with φ
end
```

Initiator Role. When a host s wants to communicate with a host t over a consent-routed path, it first runs the CONPASS protocol with t. The initiator s establishes a secure and reliable connection with the responder t and then performs the steps outlined in Algorithm 1. For this, the initiator is assumed to know an initial set of path segments $\hat{\sigma}_1, \ldots, \hat{\sigma}_k$ from s to t. In a PAN architecture, this assumption holds since hosts are provided with a set of path segments that are available.

The path segments are filtered by the initiator a first time *before* exchanging the protocol messages, mainly because the responder would otherwise not be able to know all path segments that have bilateral consent. This is especially important if the subsequent communication uses a transport protocol with support for asymmetric unidirectional flows, such as Multiflow QUIC [11], where both communicating endpoints can choose the network paths for their outgoing flows independently.

After receiving the path segments filtered by the responder, the initiator filters these path segments a second time. In most cases, the second filtering has no effect since $\sigma_{m+1}, \ldots, \sigma_{m+n}$ are usually a subset of the segments $\sigma_1, \ldots, \sigma_m$, which are accepted by the initiator. In exceptional cases, however, the responder might include new segments in its response; the initiator then needs to execute a second filtering to remove the unwanted new segments. The primary reason for this would be an information asymmetry where the responder is aware of path segments that are unknown to the initiator (Section VII).

Responder Role. In a planned consent-routed communication between s and t, the role of the responder in CONPASS could be taken either by t itself or by a host that runs the protocol on behalf of t. Delegating the responder role to a different host can be useful for many reasons, such as to reduce the latency overhead by bringing the responder closer to the initiator (responder delegation is further discussed in Sections IV-C and VII). Once a secure and reliable connection is established, the responder runs as described in Algorithm 2.

The only task of the responder is to filter the path segments received in the initiator's REQUEST message, and to respond with the segments that also have the responder's consent. The responder is in principle allowed to furthermore include new segments in its response, yet this is only useful if some path segments are known to the responder but not to the initiator.

Algorithm 2: Consent negotiation from the responder's perspective. φ is the responder's path segment filter.

Data: φ
begin
  get REQUEST from initiator according to Eq. 1
  $\sigma_{m+1}, \ldots, \sigma_{m+n} \leftarrow$ filter $\sigma_1, \ldots, \sigma_m$ with φ
  send RESPONSE to initiator according to Eq. 2
end

Consent Logic. Both protocol participants are free to use any kind of internal logic for giving consent to certain path segments and for denying it to others. We suggest that trust assumptions should be taken into account, but many other factors can be considered when making this decision; participants can, amongst others, incorporate territorial considerations, e.g., data protection laws/surveillance apparatus in different countries, or knowledge about the security posture of ISPs, e.g., excluding ISPs whose routers have been more vulnerable to attacks.

Even though the consent logic is arbitrary, we can distinguish two fundamental approaches: taking into account either local or global path properties. This distinction is useful because the number and size of the exchanged path segments heavily depend on the approach. Examples of *local* path properties are the trust in a particular network node, the maximum bandwidth of a link, or the MTU on a path segment. These are properties that can directly disqualify a path segment. If a host's consent logic is exclusively based on local path properties, it can filter all path segments by individually dropping those who do not satisfy the requirements. Hence, filtering based on local path properties is very efficient.

On the other hand, for *global* path properties it matters how the path segments are composed into end-to-end paths. Examples are the minimum RTT, the path length in number of hops, or allowed hop sequences. If such a property is part of a host's consent logic, it is in general not possible to give consent to individual path segments; only combinations of segments into end-to-end paths can be validated against the requirements. In such a case, it is necessary to enumerate all possible end-to-end paths from the given path segments (using segment compositions as an efficient representation), and to then filter these paths according to the consent logic.

With this in mind, a segment filter taking as input a set of segments Σ might not just output a subset of Σ. In general, it outputs a subset of Σ^*, of all allowed segment compositions using segments from Σ.

Path Enumeration. In the following cases, the enumeration of end-to-end paths between s and t from a set of path segments is necessary: before filtering—if the host's consent logic takes into account global path properties—and after the protocol run, when the initiator wants to create a forwarding path for the planned communication, based on the bilaterally accepted path segments. In both cases, the procedure is the same and follows a small set of simple rules:

1) Every valid end-to-end path starts with s and ends with t, hence the first segment in a path must start with s and the last segment must end with t.
2) Two path segments σ_1, σ_2 can only be concatenated if they are adjacent, that is, if the last hop of σ_1 is the same as the first hop of σ_2.
3) Paths must not contain loops. More specifically, every network hop may appear at most once in a path.

If G is a directed multigraph in which every edge (u, v) corresponds to a path segment in the network topology with first hop u and last hop v, then the path enumeration problem from above is reduced to the problem of compiling s-t simple paths in G, a problem that has been studied before [41]. While path enumeration is a computationally hard problem in general—the number of end-to-end paths is potentially exponential in the number of segments—it can be solved efficiently in certain PAN architectures. In SCION, for example, there is a limited number of path segments and end-to-end paths consist of at most three successive segments [29], which yields a cubic number of end-to-end paths and a cubic runtime complexity in the worst case. Path-aware networks that do not have such a limit for the length of end-to-end paths should implement appropriate safeguards as discussed in Section V.

C. Protocol Extensions

While the CONPASS protocol is very simple at its core, one of the protocol's objectives is also to provide extensibility features that enable new use cases. This extensibility is achieved through two types of option fields: *per-message* option fields, included once in the message header, and *per-segment* option fields, included in every segment description. These two option field types are distinct but share the same format, as defined in Equation 8. For clarity, they have been omitted in the previous protocol message definitions; now we redefine the protocol messages with per-message options,

$$\text{REQUEST} := (s \parallel t \parallel \text{OPTS} \parallel \sigma_1 \parallel \cdots \parallel \sigma_m), \tag{5}$$
$$\text{RESPONSE} := (\text{OPTS} \parallel \sigma_{m+1} \parallel \cdots \parallel \sigma_{m+n}), \tag{6}$$

and the path segments with per-segment options,

$$\sigma := (\text{TYPE} \parallel \text{CONSENT} \parallel \text{OPTS} \parallel \text{VAL}_1 \parallel \cdots \parallel \text{VAL}_\ell), \tag{7}$$

where each option consists of a code and an option payload,

$$\text{OPTS} := (\text{CODE}_1 \parallel \text{OPT}_1 \parallel \cdots \parallel \text{CODE}_k \parallel \text{OPT}_k). \tag{8}$$

For compatibility reasons, hosts should simply ignore the option payload after an unknown option code.

In the remainder of this section, we discuss possible CONPASS extensions and their use cases.

Error Signaling. The base protocol does not foresee a mechanism for the responder to signal errors to the initiator, for example due to malformed segments in the REQUEST message. Such a mechanism can easily be implemented through a per-message option, where the option payload contains an error code as well as an optional error message.

Responder Delegation. If the CONPASS protocol is run to establish a consent-routed forwarding path between two hosts s and t, the responder can be t itself but it can also be a designated CONPASS server that runs the protocol on behalf of t (and other hosts). Such servers could be deployed at the Internet edge and run CONPASS as a service with the consent logic of the host that is targeted by the initiator.

The problem that could be solved with a protocol extension is how the delegated responder knows which destination the initiator wants to connect with and whose consent logic needs to be applied. Although t is specified in the REQUEST message, its value is only meaningful on the network-layer (e.g., for inter-domain routing, s and t would represent ASes and not individual hosts). A per-message option field in the REQUEST message can specify more precise information like the target IP address and even the target service's port number.

Segment Metadata. If the underlying network provides end hosts not only with path segments but also with some metadata about these segments, like minimum latency or maximum bandwidth, then the per-segment option fields can be used by the initiator to transmit these metadata to the responder. The metadata would allow the responder to apply a more sophisticated consent logic, also accepting or denying segments based on other properties than the mere hop sequence.

Network Capabilities. For every accepted path segment, the responder could add a "proof-of-consent" signature or MAC, as a cryptographic attestation that the initiator is currently authorized to send packets from source s to destination t using this path segment. In a capability-based network architecture [39], t would check this short-term authorization, stamped on the packet by s, and drop packets that were sent over an unauthorized network path. Such a system can be used to enforce the receiver's consent.

Segment Prioritization. In the current CONPASS protocol, routing decisions are binary—in the sense that a path segment either has the host's consent or it does not. While this information is essential in the current definition of consent routing, protocol participants may want to further prioritize the accepted path segments. Using a per-segment option, it is possible to specify a utility value for individual path segments from which the utility of an end-to-end path can be derived.

V. SECURITY ANALYSIS

Adversary Model. For the security analysis of the CONPASS protocol, we assume a full Dolev-Yao adversary [14]. In particular, the adversary can view, modify, drop, inject, and replay traffic on all paths between source and destination but cannot break cryptography.

Transport Security. If protocol messages are sent unencrypted and unauthenticated, they can be read, modified, and spoofed by the adversary. In particular, the adversary can manipulate the consent negotiation and thus change the routing to its own advantage. CONPASS must therefore be run over a secure transport layer, such as TLS-encrypted QUIC or TCP, and the responder must be authenticated to the initiator.

Confidentiality of Consent Logic. The adversary may want to gain insights about a host's consent logic, including sensitive information like which parts of the network are deemed trustworthy by this host.

A straightforward way for the adversary to gain such information from a CONPASS responder r is to initiate one or more protocol runs with r and observe which path segments are accepted by r. Since the trust assumptions are never shared explicitly, the adversary can only guess the responder's filtering rules based on the accepted path segments. If a host does not want to run the protocol as a responder with arbitrary and potentially adversarial initiators, it can require the initiator to authenticate itself and thus exercise access control. In a more passive attack, the adversary can use the same strategy to gain information about an initiator's logic if that host is running the protocol with the adversary.

By monitoring and analyzing (encrypted) CONPASS traffic, the adversary can also derive some information about the protocol participants' consent logic, without participating in the protocol itself. From the amount of data respectively sent by initiator and responder, the adversary can guess the percentage of available path segments that were accepted by both peers. After observing many negotiations of the same host with peers at different locations inside the network, the adversary could potentially infer which path segments are accepted by this host and which ones are not. If preventing such analysis is important, the protocol messages can be padded to a standard maximum size using a per-message option field.

Finally, in a network where consent routing is standard, a Dolev-Yao adversary can simply observe which network paths are used by which hosts, and infer that these paths are likely composed from path segments that have both communicating endpoints' consent. The ability for on-path adversaries to make such inferences is inherent to consent routing and a trade-off that needs to be accepted. In a weaker adversary model where the adversary is not present on the consent-routed paths, traffic analysis is no longer possible.

Network-Level DoS. Even though the adversary cannot read, modify, or spoof encrypted and authenticated communication in our model, it can simply prevent two protocol participants from communicating by dropping all their packets. Such an attack can only be mitigated when assuming end-host path control; in that case, the initiator restarts the protocol over a new forwarding path until a path is chosen that is not controlled by the adversary.

Application-Level DoS. Carefully crafted protocol messages can lead to a denial-of-service condition at the host receiving the message. An infinite recursion or loop could be provoked by creating cyclic dependencies through segment compositions. The simplest example is where a segment composition references itself as a subsegment. The mitigation for this is to only allow references in segment compositions to those segments that appeared before the referencing segment in the order of transmission, as described in Section IV-B. This requirement must be checked by every protocol participant when receiving a protocol message. Similarly, the implementation of the path enumeration must not assume that the directed path segments form an acyclic graph. Otherwise, even a simple path segment that starts and ends with the same hop could trigger an infinite loop or recursion at the host enumerating the paths.

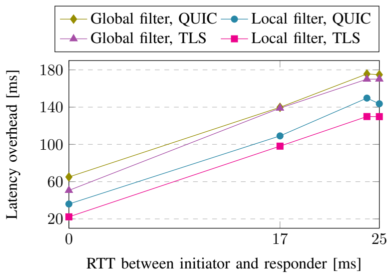

Figure 3. Average latency overhead of the CONPASS protocol, put in perspective with the RTT between the protocol initiator and responder. Four different configurations of transport protocols and path filters are compared.

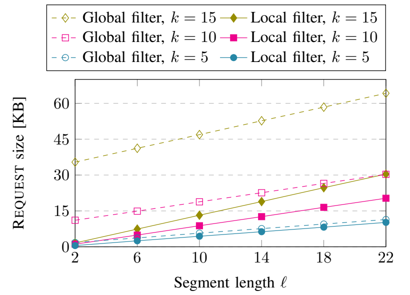

Figure 4. Worst-case size of the REQUEST message in KB as a function of the segment length ℓ, depending on the number k of segments per segment-type and the initiator's path filter.

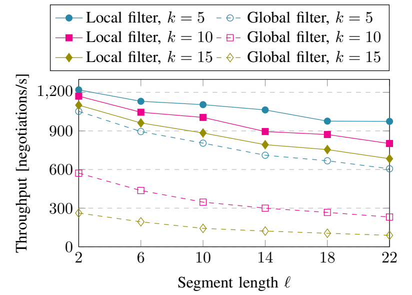

Figure 5. Worst-case CONPASS responder throughput (protocol runs per second) as a function of the segment length ℓ, for different numbers of per-type segments k and different filter configurations.

The path enumeration must also be guarded against inputs that could lead to a large computational overhead. Consider a topology: for $n \geq 1$, let $v_0, \ldots, v_n$ be $n + 1$ nodes such that $v_0 = s$, $v_n = t$, and there are two distinct segments between each pair of nodes (v_i, v_{i+1}), $i \in \{0, \ldots, n-1\}$. In this example, there are $2n$ segments that can be combined into 2^n end-to-end paths. Without any precautions, such an input would trigger exponential runtime and memory usage during path enumeration. We see two possible solutions to this problem: (a) the maximum path length n (in terms of segments) is bounded by a small constant, either by design of the PAN architecture (e.g., in SCION $n = 3$ and in NIRA $n = 2$) or through an artificial bound, and the path enumeration disregards possible end-to-end paths that would contain more than n subsegments. Or, (b) the implementation stops enumerating after a small enough number of paths; a constant number of paths can be enumerated in linear time using depth-first search.

VI. IMPLEMENTATION AND EVALUATION

In this section, we present and evaluate our CONPASS implementation (available online under https://github.com/mblarer/conpass). We measure the communication overhead of CONPASS and evaluate its scalability.

A. Implementation and Measurement Setup

To show the feasibility of consent routing in practice, we have instantiated the CONPASS protocol for one of the current path-aware network architectures and created an implementation written in Go. The network architecture of our choice is SCION since it comes with a mature and open-source codebase [1] and with real-world deployments [23], [24].

Protocol Instantiation. CONPASS is a generic protocol that needs to be adapted to the underlying PAN architecture. In particular, the values s, t, and HOP in REQUEST and RESPONSE messages (see Equations 4, 5, and 6) need to be defined in more detail. Since SCION is an inter-domain routing architecture, SCION paths contain AS-level hops and therefore, s, t, and HOP represent SCION ASes. More precisely, each hop consists of a four-tuple (ISD, AS, INGRESS, EGRESS), where ISD is the identifier of the hop's SCION isolation domain, AS is the hop's AS identifier, and INGRESS/EGRESS represent the border routers' identifiers at the ingress/egress points of the AS. Since s is the source AS, it does not specify INGRESS, and the destination AS t does not specify EGRESS. The same holds for the first and last hops of every segment.

Consent Logic. Our implementation allows applications to define arbitrary consent logic for the filtering of path segments but also integrates with SCION's path policy language [35]. For the evaluation, we compare two representative segment filters whose runtime complexity for filtering one path segment is linear in the number of hops. The "local filter" is based on a local path property, a denylist of untrusted hops, and does not require path enumeration; the "global filter" is based on a global path property, enforcing a certain sequence of hops, and therefore requires enumerating end-to-end paths, as explained in Section IV-B. Clearly, the user-defined consent logic could be arbitrarily more complex than the filters in our evaluation; however, filters with linear complexity in the number of hops per segment should be sufficient in most cases.

Message Transport. Our current implementation is compatible with any transport mechanism that provides a reliable, in-order byte stream. In particular, we can use the CONPASS protocol to negotiate consent for SCION paths over an IP connection. For security reasons, CONPASS should run over a secure transport protocol. In our evaluation, we therefore use the `quic-go` package for QUIC [9], [20] and `crypto/tls` from the Go standard library for TLS over TCP [31].

Measurement Setup. All measurements were done on a commodity machine with 16 GB RAM and an Intel Core i7 1.80 GHz quad-core processor (8 cores with hyperthreading).

B. Latency Overhead

The latency overhead of CONPASS is the additional delay until a new connection can be established over a consent-routed path, compared to the delay of plain SCION. In order to realistically evaluate this latency overhead, we have deployed CONPASS on three AWS virtual machines (in the responder role), each at a different data center location, as well as on our local testing machine (initiator and responder roles); this setup is a realistic reproduction of a CDN-based approach in which the consent negotiation is brought closer to the initiator through responder delegation. The RTT over IP between the CONPASS initiator on our machine and the different responders is 0 ms locally, and 17/24/25 ms for the three AWS VMs. All measurements have been done with SCION path segments taken from the SCIONLab testbed [24].

Figure 3 shows the latency overhead of the consent negotiation with four different configurations, as a function of the RTT between CONPASS initiator and responder. The overhead is measured separately for QUIC and TLS (over TCP). The other part of the configuration is whether initiator and responder use the local filter or the global filter. The measurements are averaged using segments from the same source AS to 27 different destination ASes in the SCIONLab topology [2] and over six repetitions per destination AS, i.e., over more than 160 runs in total.

For RTT values up to 25 ms, we measure an average latency overhead of 20–180 ms. The benchmarks also show that the actual processing overhead (i.e., the overhead when initiator and responder run on the same host with zero RTT) is only 20–65 ms. The remaining delay of up to 115 ms is incurred by the transport layer (including TCP and TLS handshakes; QUIC also uses the TLS 1.3 handshake) and therefore depends on both the transport protocol and the RTT between initiator and responder. Our implementation currently does not use TLS' 0-RTT session resumption feature, which could further reduce the latency in a real-life deployment. Section VII discusses possible strategies to deal with this latency overhead.

C. Traffic Overhead and Scalability

We now evaluate the overhead of additional traffic generated by the CONPASS protocol, as well as the scalability of our implementation, with respect to the number of available path segments and the number of hops in each segment, and assuming the worst network topology possible. The topology represents an inter-domain network where hops are ASes.

Worst-Case Network Topology. In our worst-case network topology, a fixed number of path segments yields a maximum number of paths when enumerating the end-to-end paths, as occurs when applying a path filter based on global path properties. For the following measurements, we consider the SCION's path length, $n = 3$ [29]. To reproduce the business relationships of today's Internet, we assume that there are three segment types: "up"-segments starting at the source AS, where all pairs of successive ASes along the segments are in a customer-provider relationship, "core"-segments, where consecutive ASes are in a peering relationship at the core of the Internet, and "down"-segments to the destination AS, where ASes are in a provider-customer relationship. To parametrize our topology, we define k as the number of segments of each type, and ℓ as the number of AS-level hops on each path segment. In total, the network topology contains $3k$ path segments, from which k^3 end-to-end paths can be constructed. Different architectures have different limits for the number and length of segments that are provided to end hosts; in SCION, end hosts are provided with up to $k = 10$ segments of each type, and end-to-end paths are at most 64 AS-level hops long [36]. The length of an end-to-end path is $3\ell - 2$, therefore we consider path segments of length $\ell \leq 22$ hops.

Table I

WORST-CASE SIZE OF THE RESPONSE MESSAGE IN KB, FOR DIFFERENT NUMBERS OF SEGMENTS k PER SEGMENT-TYPE, AND FOR DIFFERENT PATH FILTER TYPES OF THE INITIATOR (I) AND RESPONDER (R). THE RESPONSE SIZE IS INDEPENDENT OF THE SEGMENT LENGTH ℓ.

RESPONSE size [KB]	$k = 5$	$k = 10$	$k = 15$
I: local filter, R: local filter	0.11	0.20	0.29
I: local filter, R: global filter	1.27	10.0	33.8
I: global filter	0.77	6.02	20.3

Traffic Overhead. Consent routing, if achieved through the CONPASS protocol, does not limit the goodput of actual data flows since the consent negotiation happens before the communication, and the packet structure is unchanged. What we call traffic overhead here is the amount of data that is exchanged between a CONPASS initiator and responder.

Figure 4 shows how the worst-case size of a REQUEST message depends on the length of individual segments, on the number of segments, and on whether the client filters the path segments based on local or global properties. In the latter case, the client enumerates all possible end-to-end paths, which leads to an increased overhead, especially with growing k. The worst-case RESPONSE sizes, depending on k and on the path filters used, are given in Table I. These sizes are independent of the segment length ℓ since the responder can reference segments from the request in the response instead of repeating every segment and its hops.

Even with $k = 15$, i.e., a larger value for k than what is used in SCION, the added sizes of the REQUEST and RESPONSE messages do not exceed 85 KB. In comparison, according to the HTTP Archive page weight report [5], half of the websites in September 2021 required browsers to download more than 2100 KB of resources (e.g., HTML, CSS, JavaScript, images) with an empty cache, and over 98 percent of the websites required more than 85 KB. We can conclude that in most cases, the consent negotiation on a network with $k = 15$ generates less traffic than visiting a website for the first time, and less than 1/24 of the traffic in the case of a medium-sized website.

Scalability. We also evaluate how well the consent negotiation scales with larger networks, in terms of the number k of segments per segment type, and in terms of the segment length ℓ, which corresponds to the number of hops per segment. To this end, we measure the worst-case throughput of the CONPASS responder in answered requests per second under these various conditions, also repeating the measurements for filters that require path enumeration (global filters) and filters that do not (local filters). All benchmarks were done on our local test machine with 8 parallel threads for the responder. The results of these measurements are shown in Figure 5.

The throughput measurements at the CONPASS responder indicate that the consent negotiation scales well with the segment length ℓ. For all measured configurations, an elevenfold increase in the segment length (from 2 to 22) reduces the throughput by a factor less than 3. The negotiation also scales with respect to the number k of segments per segment type. In the benchmarks with filters that do not require path enumeration, a three-fold increase of k (from 5 to 15) reduces the throughput by less than 1.6 times. In the benchmarks where path enumeration is required by the filters, the same increase of k leads to 27 times more end-to-end paths and to a throughput reduction by a factor less than 7.

An important insight from these measurements is that the properties on which segment filters rely make a significant difference in the performance of the system. Global properties such as path latency or path length require all possible end-to-end paths to be constructed and lower the performance of the system. Whenever possible, the path filters should therefore be limited to using only local path properties such as maximum bandwidth, or trust in particular ASes.

VII. Discussion

Management of Latency Overhead. It is possible and recommended to cache the negotiated paths after every consent negotiation. This eliminates the need to renegotiate later when connecting with the same host again. However, when two hosts connect for the first time and the CONPASS protocol is run beforehand, the start of the communication is delayed by up to a few hundred milliseconds, depending on the RTT between CONPASS initiator and responder. According to the use case, there are different ways to deal with this latency overhead:

- In the following cases, the latency overhead can simply be *accepted*: (i) if the additional overhead of the CONPASS protocol is negligible compared to the total duration of the communication (e.g., large file transfers, video streaming, VoIP calls), (ii) if the protocol responder knows better how to choose good paths for the given service or application (e.g., a web service where large-bandwidth paths are more beneficial than low-latency paths), (iii) if the bilateral trust in the communication is more important than the fast connection setup.
- If connections with new and unknown hosts need to be established as fast as possible (e.g., in web browsing) and the first contact can be done over a path solely chosen by the sender, the overhead can be *avoided* as follows: the communication and the consent negotiation are started in parallel; once the consent negotiation is finished, the communication is moved to a consent-routed path.
- If the actual communication is required to start on a consent-routed path but the latency overhead for that is deemed too high, the remaining option is to *reduce* the overhead. Since the RTT between a CONPASS initiator and responder has a big impact on the latency overhead, the most viable strategy is to move the CONPASS responder closer to the initiator. This could be done following an edge-computing approach where the target host delegates the consent negotiation to a dedicated and replicated CONPASS server using the protocol extension for responder delegation (see Section IV-C).

Resolution of Consent Disagreement. Consent routing is a meaningful concept as long as there is at least one path that has the consent of both communicating hosts. If such a path is not found after running CONPASS, different use cases might favor different resolution strategies: if the initiator can ease its constraints, it may re-run the protocol with more accepted path segments this time; or, if the responder is not strict about its consent logic, it may always accept at least one path that is "as acceptable as possible". If both hosts see more value in adhering to their path requirements than in communicating, they may as well decide not to connect at all.

Recovery from Path Failures. Routing mechanisms are expected to be resilient to path failures, for example, due to broken links or offline routers. In a PAN architecture, hosts can immediately switch to a new path once the path failure has been detected. In order to enable the same with consent-routed paths, CONPASS allows hosts to find not just one end-to-end path that respects their respective consent logic but all n paths that do so. This means that after one of these paths has failed, there are potentially $n-1$ other consent-routed paths that can replace it. If all negotiated paths happen to fail at once, hosts can use the same resolution strategy as in the case of consent disagreement and renegotiate if needed.

Hidden Paths. In some cases, the CONPASS responder might be aware of path segments that are unknown to the initiator, and include them in its response. In SCION, so-called *hidden paths* are made from such path segments and used as a DoS defense on the network level. More precisely, hidden paths in SCION contain down-segments that are not publicly registered by the destination AS and that require packets to be specially authenticated. Unauthenticated packets are dropped before even reaching the destination. One challenge of hidden paths in SCION is their distribution to end hosts. If the responder is part of the AS infrastructure and reachable through a normal SCION path, the CONPASS protocol can serve as a way to solve this distribution problem. The hidden path can be included in the CONPASS response, and the authenticators can be added through a dedicated per-segment option field.

CONPASS over VPN or TLS. TLS encryption is robust, but sophisticated (e.g., nation-state) attackers can perform traffic analysis on encrypted communication, analyzing communication time, frequency, and size. VPN is also vulnerable to blind in/on-path attackers, who can infer communication over VPN and inject packets [37]. TLS and VPNs benefit from CONPASS, which enforces packet routing over the most trusted paths, avoids in/on-path attackers, and complicates traffic analysis.

Traffic Engineering. The CONPASS protocol allows a host to adapt its consent logic to the host with whom it is negotiating and can therefore be used for traffic engineering. If a server is experiencing a large number of incoming connections, it can use the protocol to steer different hosts to different network paths and hence do load-balancing on the network. Moreover, a server with different types of customers can provide each customer type with a different set of paths. For example, paying customers could get paths with lower latency or higher bandwidth than non-paying customers.

Responder Delegation and Discovery. Section IV-C describes a CONPASS extension that can be used by responders to run the consent negotiation on behalf of other hosts. This responder delegation allows for more flexible and efficient deployments; for example, a company hosts a designated CONPASS server in its network that acts as the delegated responder for all company servers; or, a global web service delegates the consent negotiation to a trusted content delivery network (CDN), which is deployed at the edge of the Internet and whose proximity with potential clients reduces the latency overhead for the negotiation. While the CONPASS extension solves the problem of informing the delegated responder about the service that the initiator wants to connect with, two additional challenges need to be addressed: (i) how the target service delegates the responder role and (ii) how the initiator knows where to find the delegated responder.

Depending on the use case, the arrangement for setting up responder delegation could be made differently. If the delegated responder is under the same administrative control as the target host, a network administrator can do the configuration work. Otherwise, responder delegation could also be offered as a service by cloud providers or CDNs and purchased online. Apart from any contractual work, setting up responder delegation requires the target host or service to authenticate itself to the authority behind the delegated responder, similar to how hosts authenticate themselves to a certification authority with the ACME protocol [6]. This prevents malicious actors from delegating consent negotiation for destination addresses that they do not own. The second part of the setup is the registration of the target service's consent logic at the delegated responder, for example through a domain-specific language [35] in which common constraints and requirements for path segments can be expressed.

A host wishing to negotiate paths for a connection with host t should be able to learn the delegated responder's network address by the same means as it learns t's network address. If it is through name resolution, then the DNS SRV record [18] can be used to register and resolve the address of the delegated CONPASS responder for a given domain.

Incremental Deployment. CONPASS is designed as an application-layer protocol for hosts in path-aware networks. As such, it can be deployed without any changes to the underlying networking architecture. The protocol is more likely to be used as part of applications that benefit from particular paths rather than as a separate service because every application might use CONPASS in a different way, depending on the use case. In a path-aware network architecture, such applications can leverage the consent negotiation to offer users more control over the paths on which they want to receive data. Server applications can benefit from the possibility to do traffic engineering by offering different paths to different clients, which balances the load on different network paths and eventually also benefits the clients. Overall, path-aware applications that adopt the CONPASS protocol are more attractive for their users since they enable the use of communication paths that are bilaterally trusted and benefit both communicating parties.

Future Work. Finding more secure, efficient, and privacy-preserving ways to achieve consent routing remains an open research problem. This paper proposes a first approach where two communicating hosts in a path-aware network negotiate a set of bilaterally accepted path segments. Destination hosts can delegate this negotiation to a third party, but this currently requires them to share their consent logic with the third party. Trusted execution environments such as Intel SGX-based enclaves [4] could further improve this approach by allowing hosts to delegate the consent negotiation to the third party without revealing their consent logic.

VIII. RELATED WORK

BGP routes inter-domain traffic between autonomous systems in the backbone of today's Internet. Despite its prevalent use to this day, BGP is lacking adequate security mechanisms, which makes it prone to BGP hijacking attacks like SICO [7], and other attacks that rely on BGP poisoning, such as the Maestro attack [27]. There have been various attempts to improve the security of BGP, including S-BGP [22], soBGP [28], SPV [19], and BGPsec [25], none of which has seen wide deployment. Other research has focused on improving the performance of BGP-based inter-domain routing through performance-aware traffic engineering architectures such as Espresso [40], Edge Fabric [32], [33], and ARROW [43].

As an alternative to BGP's dynamic routing, path-aware network architectures have emerged, including Platypus [30], NIRA [38], Pathlet Routing [17], and SCION [42]. While such architectures can fundamentally solve many of the security problems that exist with BGP routing, they can also achieve high efficiency and allow packet senders to select forwarding paths. Route Bazaar [8] applies ideas from path-aware networking but remains backwards-compatible using BGP.

The concept of consent routing, presented in this paper, is in line with the aforementioned efforts to improve the security of Internet routing and its utility to end hosts. We build upon this idea but go one step further: from unilateral path decisions made by the sender, to bilateral path decisions made by the sender *and* the receiver. Thus we expand the benefits of end-host path selection to both communicating hosts, offering the ability to receive packets over trusted forwarding paths.

IX. CONCLUSION

Path-aware network architectures have demonstrated their ability to increase both the security and the efficiency of the Internet with end-host path selection. A major benefit of enabling path control for end hosts is that they can choose the safest and most appropriate route for packets according to their trust assumptions and needs. However, in the path-aware architectures proposed so far, this is only true for the sender of a packet. The receiver's trust assumptions and needs are still not taken into account and may be violated by the sender's path selection. The consent routing paradigm proposed in this paper addresses this shortcoming and suggests that packet senders and receivers should both be part of the routing process. With our design and implementation of the CONPASS protocol, we have demonstrated that consent routing on top of a path-aware Internet is feasible without modifications to the underlying architecture. The CONPASS protocol allows hosts to exchange bilaterally acceptable forwarding paths—it opens up new possibilities such as host-based traffic engineering on the public Internet—and paves the way for more trust and acceptance in future Internet communication.

ACKNOWLEDGMENTS

We would like to thank Patrick Bamert for his inspriation on this study; Massimo Marelli, Cedric Maire, and Justinas Sukaitis for their insightful feedback and suggestions. We gratefully acknowledge support from ETH Zurich, and from the Zurich Information Security and Privacy Center (ZISC). This work was also supported by the ETH4D and EPFL EssentialTech Centre Humanitarian Action Challenge Grant.

REFERENCES

[1] SCION Internet Architecture [GitHub Repository]. https://github.com/scionproto/scion.

[2] SCIONLab. https://www.scionlab.org.

[3] Internet Protocol. RFC 791, 1981.

[4] Ittai Anati, Shay Gueron, Simon P. Johnson, and Vincent R. Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *Proceedings of the International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[5] HTTP Archive. Page Weight Report September 2021. https://httparchive.org/reports/page-weight?start=2021_09_01&view=list.

[6] Richard Barnes, Jacob Hoffman-Andrews, Daniel McCarney, and James Kasten. Automatic Certificate Management Environment (ACME). RFC 8555, 2019.

[7] Henry Birge-Lee, Liang Wang, Jennifer Rexford, and Prateek Mittal. SICO: Surgical Interception Attacks by Manipulating BGP Communities. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[8] Ignacio Castro, Aurojit Panda, Barath Raghavan, Scott Shenker, and Sergey Gorinsky. Route Bazaar: Automatic Interdomain Contract Negotiation. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*, 2015.

[9] Lucas Clemente. A QUIC Implementation in Pure Go [GitHub Repository]. https://github.com/lucas-clemente/quic-go.

[10] Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner. The Hijackers Guide To The Galaxy: Off-Path Taking Over Internet Resources. In *Proceedings of the USENIX Security Symposium*, 2021.

[11] Quentin De Coninck and Olivier Bonaventure. Multiflow QUIC: A Generic Multipath Transport Protocol. *IEEE Communications Magazine*, 59(5):108–113, 2021.

[12] Steve E. Deering and Bob Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 8200, 2017.

[13] Roy C. Dixon and Daniel A. Pitt. Addressing, Bridging, and Source Routing (LAN Interconnection). *IEEE Network*, 2(1):25–32, 1988.

[14] Danny Dolev and Andrew Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[15] Deborah Estrin, Tony Li, Yakov Rekhter, Kannan Varadhan, and Daniel M. A. Zappala. Source Demand Routing: Packet Format and Forwarding Specification (Version 1). RFC 1940, 1996.

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

[17] P. Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet Routing. In *Proceedings of the ACM SIGCOMM Conference*, 2009.

[18] Arnt Gulbrandsen and Levon Esibov. A DNS RR for Specifying the Location of Services (DNS SRV). RFC 2782, 2000.

[19] Yih-Chun Hu, Adrian Perrig, and Marvin Sirbu. SPV: Secure Path Vector Routing for Securing BGP. In *Proceedings of the ACM SIGCOMM Conference*, 2004.

[21] David B. Johnson and David A. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks*, pages 153–181. Mobile Computing, 1996.

[20] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, 2021.

[22] Stephen Kent, Charles Lynn, and Karen Seo. Secure Border Gateway Protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, 2000.

[23] Cyrill Krähenbühl, Seyedali Tabaeiaghdaei, Christelle Gloor, Jonghoon Kwon, Adrian Perrig, David Hausheer, and Dominik Roos. Deployment and Scalability of an Inter-Domain Multi-Path Routing Infrastructure. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2021.

[24] Jonghoon Kwon, Juan A García-Pardo, Markus Legner, François Wirz, Matthias Frei, David Hausheer, and Adrian Perrig. SCIONLab: A Next-Generation Internet Testbed. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, 2020.

[25] Matt Lepinski and Kotikalapudi Sriram. BGPsec Protocol Specification. RFC 8205, 2017.

[26] Massimo Marelli. Hacking humanitarians: Defining the cyber perimeter and developing a cyber security strategy for international humanitarian organizations in digital transformation. *International Review of the Red Cross*, 102(913):367–387, 2020.

[27] Tyler McDaniel, Jared M. Smith, and Max Schuchard. The Maestro Attack: Orchestrating Malicious Flows with BGP. In *Proceedings of the EAI International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2020.

[28] James Ng. Extensions to BGP to Support Secure Origin BGP (soBGP). Internet-Draft draft-ng-sobgp-bgp-extensions-02, IETF, 2004.

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

[30] Barath Raghavan and Alex C. Snoeren. A System for Authenticated Policy-Compliant Routing. In *Proceedings of the ACM SIGCOMM Conference*, 2004.

[31] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, 2018.

[32] Brandon Schlinker, Italo Cunha, Yi-Ching Chiu, Srikanth Sundaresan, and Ethan Katz-Bassett. Internet Performance from Facebook's Edge. In *Proceedings of the Internet Measurement Conference (IMC)*, 2019.

[33] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *Proceedings of the ACM SIGCOMM Conference*, 2017.

[34] Carl A. Sunshine. Source Routing in Computer Networks. *ACM SIGCOMM Computer Communication Review*, 7(1):29–33, 1977.

[35] Anapaya Systems. Path Policy Language Design. https://scion.docs.anapaya.net/en/latest/PathPolicy.html.

[36] Anapaya Systems. SCION Header Specification. https://scion.docs.anapaya.net/en/latest/protocols/scion-header.html.

[37] William J Tolley, Beau Kujath, Mohammad Taha Khan, Narseo Vallina-Rodriguez, and Jedidiah R Crandall. Blind In/On-Path Attacks and Applications to VPNs. In *Proceedings of the USENIX Security Symposium*, 2021.

[38] Xiaowei Yang, David Clark, and Arthur W. Berger. NIRA: A New Inter-Domain Routing Architecture. *IEEE/ACM Transactions on Networking*, 15(4):775–788, 2007.

[39] Xiaowei Yang, David Wetherall, and Thomas Anderson. A DoS-Limiting Network Architecture. *ACM SIGCOMM Computer Communication Review*, 35(4):241–252, 2005.

[40] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *Proceedings of the ACM SIGCOMM Conference*, 2017.

[41] Norihito Yasuda, Teruji Sugaya, and Shin-Ichi Minato. Fast Compilation of s-t Paths on a Graph for Counting and Enumeration. In *Proceedings of the International Workshop on Advanced Methodologies for Bayesian Networks*, 2017.

[42] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G. Andersen. SCION: Scalability, Control, and Isolation on Next-Generation Networks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2011.

[43] Zhizhen Zhong, Manya Ghobadi, Alaa Khaddaj, Jonathan Leach, Yiting Xia, and Ying Zhang. ARROW: Restoration-Aware Traffic Engineering. In *Proceedings of the ACM SIGCOMM Conference*, 2021.