Hummingbird: Dynamic Path Validation with Hidden Equal-Probability Sampling

Anxiao He, Xiang Li, Jiandong Fu, Haoyu Hu, Kai Bu* *Member, IEEE,* Chenlu Miao and Kui Ren *Fellow, IEEE*

***Abstract*—Path validation has already been incrementally deployed in the Internet architecture. It secures packet forwarding by enabling end hosts to negotiate specific forwarding paths and enforcing on-path routers to prove their forwarding behaviors along these paths. Most existing path validation solutions target static paths, paying less attention to fully dynamic paths that support flexible routing. In this paper, we present Hummingbird as the first validation solution over fully dynamic paths. It features a hidden equal-probability sampling technique. Gaining efficiency via routers probabilistically sampling packets to validate, we craft the sampling probability such that each router validates a similar amount of packets given an unknown path length. We further hide the state of whether a packet has been sampled and validated using a lightweight, non-cryptographic scheme. This prevents attackers from differentiating and selectively mis-forwarding packets. We validate security and efficiency of Hummingbird through both theoretical proof and experimental evaluation.**

***Index Terms*—Path validation, dynamic routing, hidden equal-probability sampling.**

I. Introduction

As a fundamental feature for future secure Internet [1]–[3], path validation has already been incrementally deployed [4]. It enforces packet forwarding along designated paths and verifies whether packets actually follow the paths meanwhile [5]. In comparison with traditional forwarding that is transparent to end hosts, path validation can effectively mitigate various attacks such as re-routing [6], BGP hijacking [7], and packet tampering [8], [9]. There have a series of initiative path-aware Internet architectures to advocate path validation—NIRA [10], NEBULA [11], and SCION [1], [2]. The essential design strategy is to require that on-path routers add their proofs in packet headers [5], [12]–[19]. Then routers can use such proofs to validate packet forwarding, that is, which routers a packet has traversed and in what order. Recently, SCIONLab [4] has deployed various routers supporting path validation around the globe. It provides an absolute stimulus for adoption and practise of path validation.

A. He, J. Fu, and K. Bu* are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China, and are also with ZJU-Hangzhou Global Scientific and Technological Innovation Center, Hangzhou 311215, China. E-mail: {zjuhax, jiandongfu, kaibu kuiren}@zju.edu.cn

X. Li is with the School of Cyberspace Science, Harbin Institute of Technology, Harbin 150001, China. E-mail: 21S003076@stu.hit.edu.cn

H. Hu and C. Miao are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China. E-mail: {3180103843, clmiao}@zju.edu.cn

K. Ren is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China, and is also with the Zhejiang Provincial Key Laboratory of Blockchain and Cyberspace Governance, Hangzhou 310027, China. E-mail: kuiren@zju.edu.cn

*Corresponding Author: Kai Bu.

EDICS: NET-SPRO Security protocols.

However, existing path validation solutions focus mostly on static paths and do not apply to fully dynamic paths. Static path validation solutions (e.g., ICING [5], OPT [12] and EPIC [14]) drop packets once they discover route deviation. This limits their agility to adjust forwarding paths when, for example, network congestion or malicious routers need be avoided [20]–[24]. It is practically necessary to investigate how to enforce path validation while dynamically diverting packets among multiple allowed paths. A fully dynamic path raises two specific challenges for path validation. First, routers can flexibly adjust routing policies according to real-time network status [20]–[24]. This brings more freedom for routers to forward packets and offers them more chances to respond more quickly upon unforeseen situations where the network topology changes. Second, given the possibility of switching to different forwarding paths, it is even hard to predict the exact length of the actually taken forwarding path from the source to the destination. Both challenges of unknown path and length render existing solutions infeasible for dynamic path validation.

In this paper, we take on the challenge and present Hummingbird as the first dynamic path validation solution. Inspired by PPV [13], we use probabilistic sampling toward efficient validation of packets over fully dynamic paths. However, the sampling scheme in PPV requires a known path length as a critical parameter that achieves secure sampling. We leverage the reservoir sampling technique [25] to design an equal-probability sampling over a dynamic path with unknown path length. Specifically, it guarantees that the probability of a packet being sampled by a router yet not by any of its downstream routers be identical. Only with such an equal probability can we guarantee that attackers cannot select a router with a low probability of sampling to selectively attack. Furthermore, we design a lightweight non-cryptographic scheme to protect the sampling indicator in packet headers. Again, such a hidden sample state prevents selective attacks from distinguishing whether a packet is sampled and will be verified in the next hop.

We implement Hummingbird as well as a non-sampling Baseline using the Click router [26]. Hummingbird outperforms Baseline in terms of packet processing speed and network throughput, being more efficient as path length increases. To better understand the efficiency scale of Hummingbird in a well accepted context, we also use state-of-the-art static path validation solutions—PPV [13] and EPIC [14]—as references. First, the overhead of Hummingbird header fields is 101 bytes irrespective with path length; while the state-of-the-art sampling based solution—PPV [13]—imposes a 64-byte header overhead. Second, Hummingbird can achieve a throughput of 1.48 Gbps given a single-core machine with a 10 Gbps link. This comparative to the state-of-the-art high-speed EPIC [14], which yields a throughput of 1.71 Gbps in the same experimental setting.

In summary, we make the following major contributions to path validation.

- We present Hummingbird as the first solution that can efficiently validate packet forwarding along dynamic paths with unknown path length.
- We design an equal-probability sampling technique without path length as a priori. We further explore a lightweight, non-cryptographic scheme to hide the indicator bit of whether a packet has been sampled and needs to be verified. Both strategies prevent attackers from distinguishing sampled and unsampled packets and thus prevent them from selectively attacking packet forwarding, to which the state-of-the-art probabilistic PPV is vulnerable [13].
- We implement Hummingbird using the Click router [26] and DPDK [27] and validate its security and efficiency through extensive experiments. It can preserve the hidden equal-probability sampling property and achieve a comparative throughput to the state-of-the-art path validation solutions.

The rest of the paper is organized as follows. Section II models dynamic path validation and underlines the necessity of exploring a corresponding solution. Section III accordingly presents Hummingbird as the first path validation solution that supports fully dynamic paths. Section IV details the Hummingbird design. Section V and Section VI validate security and efficiency of Hummingbird through theoretical analysis and experimental evaluation, respectively. Finally, Section VII concludes the paper and indicates future work.

II. Problem

In this section, we define the problem of dynamic path validation. We then investigate the incompatibility of existing path validation solutions to dynamic path validation.

A. System Model

Figure 1 illustrates the system model of our interest. It follows the architecture of SCIONLab [4], a deployed path validation platform in the Internet. SCIONLab deploys various routers supporting path validation around the globe. Given the deployment cost, an autonomous system (AS) may or may not be selected. Since such SCIONLab routers enforce path validation, we consider them trusted. In contrast, we consider any other routers without path validation incorporated as potentially malicious or untrusted [8], [9] (Section II-B). Figure 1 illustrates a network that consists of a minority of trusted routers and a majority of untrusted routers.

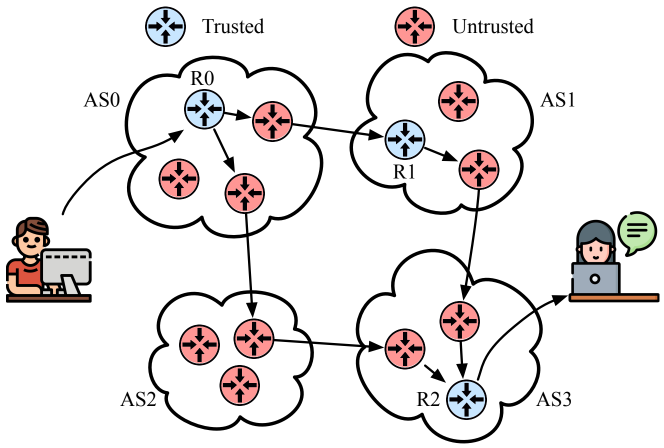

Fig. 1: System model with routers supporting dynamic path validation [4]. Routers R0, R1, and R2 incorporated with path validation functions are considered trusted.

They preserve packet integrity and forwarding correctness. In contrast, other routers may attack packet integrity and forwarding. Routers (e.g., R0) can dynamically adjust forwarding paths in real time to fit network status.

B. Adversary Model

As with existing solutions [5], [12]–[16], we consider an active attacker that succeeds if it can forge a proof to pass path validation. Specifically, any router that does not enforce path validation in Figure 1 might be a potential attacker. The ultimate goal of proof forging is to make a mis-forwarded packet deemed valid. According to the requirement of path validation, a valid packet should traverse through routers on the designated forwarding path in the correct order. Once the attacker bypasses an on-path router, redirects packets to a different path, or injects a forged packet to the forwarding path, corresponding packets should be detected and discarded. Sophisticated attackers may collude with each other to gain further attack leverage by, for example, correlating packet contents. Path validation should also be robust against such a collusion attack. We will demonstrate Hummingbird robustness against various attacks in Section V. Preserving privacy [28] is considered out of the scope.

C. Incompatibility of Existing Path Validation Solutions

Albeit dynamic path validation is comparatively critical as static path validation, we observe that existing solutions cannot effectively validate dynamic paths. Specifically, they all require a priori knowledge of the entire forwarding path. This can be easily observed from most existing solutions that target only static paths [5], [12]–[16]. For example, ICING [5] requires that each router beware the forwarding path it is on such that it can provide its own credentials to downstream routers as well as verify credentials of upstream routers toward itself. OPT [12] and follow-up solutions [14], [15], [17]–[19] improve efficiency through enabling a trusted source to precompute credentials for downstream routers. In contrast with earlier solutions, PPV [13] further boosts validation efficiency by requiring that a packet be validated by only two sampled adjacent routers. It uses a flow of packets to jointly testify the forwarding correctness along the path. Packet sampling is uniformly distributed across routers, that is, each router samples a relatively equal amount of packets so no router suffers from biased testification. This requires the exact value of path length to determine the sampling probability on each router.

Among existing solutions, Atlas [19] and PSVM [16] embrace more than one single static path yet still do not support fully dynamic paths. Atlas [19] initiates path validation for networks with multipath routing. Mutipath routing allows a forwarding path to divert. Atlas proposes a hierarchical validation technique to compress proofs of multiple paths. The key benefit is removing redundant proof fields of overlapping routers on more than one path. However, it not only targets static paths but also induces a high overhead to carry multipath proofs in packets. PSVM [16] does not encode all multipath proofs into packets. Instead, the source follows the initial routing decision to encode proofs in a segment wise way. A trusted agent from service providers called Credible Guarantee Agent (CGA) handles dynamic routing. If a forwarding path needs to divert, CGA informs the corresponding router with the new routing decision. Moreover, CGA pre-computes proofs for all segments on the new path and issues them to the diverting router, which uses these new proofs to replace the carried proofs in the received packets. Albeit promising higher flexibility and efficiency than Atlas does, PSVM still requires any adjusted path to be known prior to packet forwarding. CGA may also be vulnerable to a single point of failure in highly dynamic networks.

Note that path validation also shines forth in recent reliable forwarding solutions for Software-Defined Networking (SDN) [29]–[34]. The key difference between hardware logics on SDN switches and Internet routers is that SDN switches rely more on matching-then-forwarding [35]. Different from Internet routers, SDN switches lack sufficient computation power to directly support various operations enforced by existing path validation solutions. They thus have to resort either 1) to the controller for examining rule conflicts [29], [31], collecting and analyzing flow statistics [30], [33] or 2) to end-hosts for verifying forwarding correctness of received packets [32], [34]. In this paper, we concentrate on path validation in the Internet infrastructure, where enroute routers should directly enforce validation computation alongside packet processing.

Beyond existing path validation solutions, we take the challenge to enforce forwarding validation over fully dynamic paths. A fully dynamic path raises three specific challenges for path validation. First, routers can flexibly adjust routing policies according to real-time network status [20], [22]–[24]. For example, R0 in Figure 1 features two possible forwarding paths to bridge the two users. Second, packets may be attacked while they are traversing through untrusted routers. For example, both outgoing paths from R0 go through untrusted routers. In corporation with other enroute trusted routers (e.g., R1 and R2), R0 should use credentials agreed upon among trusted routers to create a proof of sufficient security. Third, given the possibility of switching to different forwarding paths, it is even hard to predict the exact length of the actually taken forwarding path from the source to the destination. However, existing solutions usually require a known path as a priori [5], [12]–[19] as we have discussed in Section II-C. These challenges render them infeasible for dynamic path validation.

III. Overview

In this section, we propose Hummingbird as the first take of efficient validation for fully dynamic paths. It gains efficiency via routers probabilistically sampling packets to validate. We explore two design techniques to guarantee a hidden equal-probability sampling.

- First, we craft the sampling probability of each router using unknown path length such that each on-path router validates a similar amount of packets. This makes sure that every segment of the forwarding path can be validated.
- Second, we hide the state of whether a packet has been sampled and validated using a lightweight, non-cryptographic scheme. This prevents the attacker from selectively mis-forwarding validated packets that, if not sampled and validated again, are more exploitable to evade path validation.

Inspired by PPV [13], Hummingbird adopts probabilistic sampling to strive for efficient validation of packets over fully dynamic paths. It features the following key ideas and properties.

Applicability to flexible routing. Hummingbird is the first lightweight solution that totally fits in a dynamic network without having to know the forwarding path in advance. Since the source does not need to compute assurances for all the downstream nodes based on a specified path, routers can choose their own next hop according to the flexible routing policy under real time network status. This brings more freedom for routers to forward packets and offers them more chances to respond more quickly upon unforeseen situations where the network topology changes.

Equal-probability sampling. Probabilistic sampling is originally proposed for boosting path validation efficiency [13]. It requires that only sampled packets be embedded with router proofs and verified later on. Guaranteeing an equal probability of sampling is thus critical for security. Only with an equal probability can we guarantee that attackers cannot select a router with a low probability of sampling to selectively attack. For example, an attacker may manipulate packets toward routers that unlikely sample them and evade detection. However, given an unknown path length, it is challenging to assign such sampling probabilities once for all.

We leverage the reservoir sampling technique [25] toward an equal-probability sampling over a dynamic path with unknown path length. It is a randomized algorithm that aims to select k items from a set $\mathbb{S}$ containing n items, where n is a large or unknown number. It may re-sample sampled packets. Each router has independent sampling probability. Since path validation focuses on a single packet rather than the entire flow, the setting is equivalent to the reservoir sampling when $k = 1$ with an unknown length of router sequence. We accordingly define the equal-probability sampling that Hummingbird supports as follows.

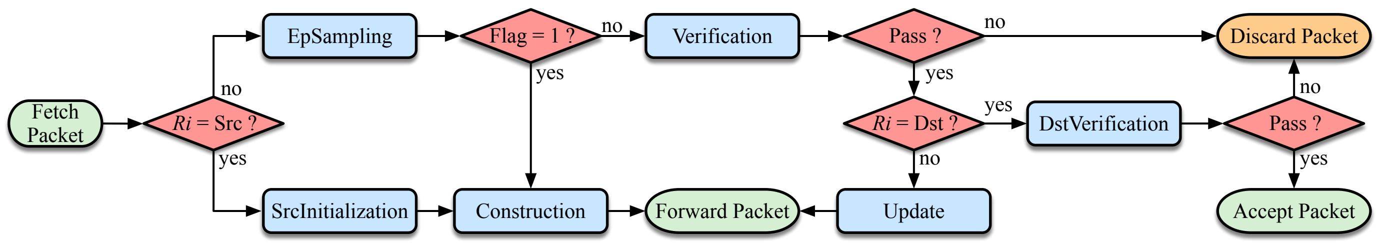

Fig. 2: Hummingbird workflow.

TABLE I: Comparison of properties that different solutions satisfy ✓, partially satisfy ✓*, and do not satisfy ✗.

Solution	Property		
	Dynamic	Equal-probability	Hidden
ICING [5]	✗	✗	✗
OPT [12]	✗	✗	✗
OSV [17], [18]	✗	✗	✗
Atomos [15]	✗	✗	✗
Atlas [19]	✗	✗	✗
EPIC [14]	✗	✗	✗
PPV [13]	✗	✓	✗
PSVM [16]	✓*	✗	✗
Hummingbird (this paper)	✓	✓	✓

Definition 1. *The probability of a packet being sampled by a router yet not by any of its downstream routers should be identical.*

Hidden sampling state. Unlike PPV that stores the sampling address in plaintext [13], we hide the packet sampling state about whether the packet is sampled and use a lightweight technique to help routers verify. Packets need a header field to indicate whether they are sampled or not [13]. Such a field helps a router to detect a sampled packet and verify its validation proof. It should be tamper-resistant to prevent strategic attacks. Otherwise, the attacker may strategically misforward only unsampled packets to avoid detection. Furthermore, the attacker may even modify it from a sampled state to an unsampled state. We protect the sample state via a lightweight keyed scheme. This prevents selective attacks from distinguishing whether a packet is sampled and will be verified in the next hop.

Table I compares Hummingbird with existing path validation solutions in terms of the three properties—applicability to flexible routing, equal-probability sampling, and hidden sampling state. Most solutions focus on static path validation. Only PSVM [16] partially satisfies applicability to dynamic networks because it still requires the information of the new path upon path switching. In contrast, our Hummingbird supports fully dynamic paths and stands out as the first solution that satisfies all the three properties.

IV. Design

In this section, we detail the Hummingbird design. We start with a high level workflow of Hummingbird functions. We then detail the Hummingbird header fields, followed by how they are constructed and verified via each function with notations and abbreviations defined in Table II. Finally, we synthesize all these functions into the Hummingbird algorithm.

TABLE II: Definition of notations and abbreviations.

Notation	Definition
Src	source node
Dst	destination node
R_i	on-path trusted router
P	the packet involving in path validation
K_{ij}	shared symmetric key between R_i and R_j
$Flag$	packet sampling state
$HiddenSample$	ciphertext of packet sampling state
$HopCount$	number of nodes the packet has passed through
$DataHash$	hash of the packet payload
$SessionID$	identifier of a session
$Timestamp$	creation time of the packet
$ValidationProof$	proof field for R_i to verify
$Concealment$	proof field for Dst to verify
$H(\cdot)$	cryptographic hash function
$\mathrm{MAC}_K(\cdot)$	message authentication code using key K
PRF	a pseudo random function used for sampling

A. Workflow

Hummingbird synthesizes six major processing functionalities on a router.

- `SrcInitialization` takes effect only on the source. It initializes Hummingbird header fields.
- `EpSampling` decides whether a router should sample a packet and add its credentials for the next hop to validate.
- `Construction` computes the aforementioned router credentials.
- `Verification` verifies whether a received packet is sampled by the previous hop and, if yes, verifies the credentials of the previous hop.
- `Update` updates Hummingbird header fields for packets that are not sampled and embedded with router credentials.
- `DstVerification` takes effect only on the destination. It verifies a packet-specific proof generated by the source.

Figure 2 illustrates the workflow of Hummingbird. We sketch how it guides packet processing on the source, intermediate, and destination routers as follows.

Source. The source first invokes `SrcInitialization` to generate packet metadata as the packet's initial state and a proof for the destination. Moreover, the equal-probability sampling design enables the source to sample every packet (Section IV-C). The source then invokes `Construction` to generate a path validation proof for the next hop to verify.

Intermediate routers. Upon receiving a packet, an intermediate router first invokes `EpSampling` to determine whether to sample the packet. If the sample-state indicator Flag is generated to be 1, the router invokes `Construction` to generate a path validation proof for the next hop to verify. Otherwise, it invokes `Verification` to verify Hummingbird header fields. If verification succeeds, the router proceeds with `Update` for updating Hummingbird header fields before forwarding the packet to the next hop.

Destination. The destination need not sample a packet since it has no next hop to prove to. Instead, the destination conducts the same verification process via `Verification` as an intermediate router and an additional `DstVerification` for verifying the proof initialized by the source. Only if a packet passes all verification throughout intermediate and destination routers can it be accepted by the destination as valid.

B. Hummingbird Header Structure

We introduce the header fields for Hummingbird to validate dynamic paths between the IP and TCP headers. This design philosophy follows the tradition of exiting path validation solutions [5], [12]–[15]. The major reason is because that validation fields and IP header fields together drive the forwarding process. Appending validation fields right after the IP header facilitates packet parsing. As shown in Figure 3, the Hummingbird header consists of seven fields. Four of them—SessionID, Timestamp, DataHash, HopCount—are not cryptographically protected; they need be computed into some or all the other three keyed fields—HiddenSample, ValidationProof, Concealment.

SessionID, Timestamp, and DataHash define packet metadata and stay constant across routers, following the design of existing solutions.

- SessionID represents the identifier of a communication session. It is synchronized by the source and destination at the begging of a session.
- Timestamp records the generation time of a packet at the source. It helps routers verify packet freshness and mitigate replay attacks [12].
- Datahash is the hash value of the packet payload. It is used for integrity check.

HopCount records the number of routers that the packet has traversed and is incremented hop by hop. It is a critical parameter for deriving equal-probability sampling albeit with unknown length of a dynamic path (Section IV-C).

ValidationProof. If a router decides to sample a packet, it computes its packet-spefic credentials into ValidationProof for the next hop router to verify (Section IV-E).

HiddenSample. Packets need a header field to indicate whether they are sampled or not [13]. Such a field helps a router to detect a sampled packet and verify its ValidationProof. It should be tamper-resistant to prevent strategic attacks (Section IV-D). Otherwise, the attacker may strategically misforward only unsampled packets to avoid detection. Furthermore, the attacker may even modify it from a sampled state to an unsampled state.

Concealment. Finally, we introduce the Concealment field for the destination to verify packet integrity in case a packet and corresponding DataHash are altered after its last verification. Specifically, Hummingbird requires that a sampled packet be

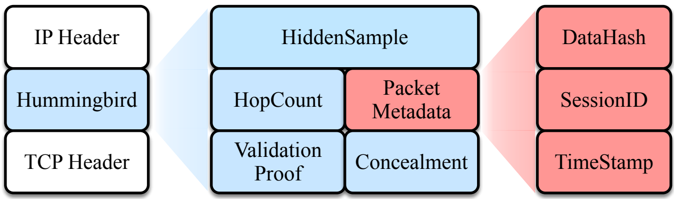

Fig. 3: Hummingbird header structure.

Algorithm 1: Equal-Probability Sampling

```
1 Function EpSampling:
2     Flag ← 0;
3     if $R_i$ is the destination then
4         break;
5     $\rho \leftarrow PRF(0, HopCount)$;
6     //$PRF$ generates a random integer $\in [0, HopCount]$;
7     if $\rho == HopCount$ then
8         Flag ← 1;
9     return Flag;
```

verified immediately on the next hop router. Any alteration over the packet can be detected upon verification of the ValidationProof field. However, if the verified packet is never sampled again, the additional Concealment field should help the destination to verify packet integrity. It is accordingly computed as a MAC value of packet metadata using the shared key of the source and destination (Section IV-H).

C. Equal-Probability Sampling

We leverage the reservoir sampling technique [25] toward an equal-probability sampling over a dynamic path with unknown path length. As Algorithm 1 shows, the sampling function `EpSampling` is effective yet simple. It takes only HopCount as the input. The output is a single-bit indicator called Flag. If `EpSampling` returns 0 for Flag, the corresponding router need not sample the packet to embed ValidationProof. Otherwise, if Flag is set as 1, the router computes and adds its credentials to the ValidationProof field (Section IV-E). The assignment of Flag depends on the value of the random integer ρ generated by PRF (line 5). Only when ρ is equal to HopCount is Flag set as 1.

Although HopCount appears to have the same functionality as the time-to-live (TTL) field that records the number of routers traversed by a packet, it cannot be replaced by TTL. The main purpose of HopCount field is acting as a parameter in Algorithm 1 to achieve the equal-probability sampling. As required by the reservoir sampling algorithm [25], sampling towards an unknown size needs a positive incremental parameter. However, TTL is a decreasing parameter. It is only applicable as a sampling parameter when the path length is known and fixed [13]. Furthermore, one may consider to choose the time that a packet receives by each enroute router as a readily available positive incremental parameter. However, it is not as stable as the hop-count number since such times of receipt can be easily affected by network congestion and packet processing speed.

Theorem 1. *Hummingbird using Algorithm 1 can guarantee the property of equal-probability sampling in Definition 1.*

Proof. Without loss of generality, assume that upon the packet reaches the destination, the full path consists of N routers with the source and destination inclusive. Let P_n denote the probability of the nth router sampling the packet. On the nth router, HopCount is equal to $n-1$. The probability of ρ being equal to $n-1$ is $1/((n-1)-0+1) = 1/n$ (line 5). Therefore, the probability of Flag being set as 1 (i.e., P_n) is equal to $1/n$ (lines 7-8). According to Definition 1, the equal probability property focuses on the probability of a packet being sampled by a router yet not by any of its downstream routers. We can accordingly calculate such probability for the nth router as follows:

$$\begin{aligned}P_n \cdot \prod_{k=n+1}^{N} (1-P_k) &= \frac{1}{n} \cdot \prod_{k=n+1}^{N} (1-\frac{1}{k}) \\ &= \frac{1}{n} \cdot \prod_{k=n+1}^{N} (\frac{k-1}{k}) \\ &= \frac{1}{n} \cdot \frac{n}{n+1} \cdot \frac{n+1}{n+2} \cdots \frac{N-1}{N} \\ &= \frac{1}{N}.\end{aligned}$$

Once the dynamic path ends up being an N-hop path, the equal sampling probability can be guaranteed as $1/N$. This proves Theorem 1. □

D. Hidden Sample State

We protect the exact value of Flag via a keyed hash—HiddenSample. The construction function of HiddenSample should satisfy the following properties:

- For the same packet, HiddenSample should vary per hop regardless of the value of Flag.
- For different packets in the same flow, the values of their HiddenSample should be different and uncorrelated on the same router even if they have identical Flag.

Both properties guarantee that the attacker cannot infer the value Flag by correlating HiddenSample of the same packet across routers or of different packets on the same router. A straightforward solution is to use a MAC using the shared key between adjacent routers (e.g., $K_{i(i+1)}$ shared by router R_i that samples the packet and its next hop router R_{i+1}) and taking Flag and DataHash as inputs—$MAC_{K_{i(i+1)}}(Flag||DataHash)$. $K_{i(i+1)}$ varies per path segment and helps satisfy the first property. DataHash varies with packet per se and helps satisfy the second property. However, we observe that we do not necessarily use computations as complex as MAC to protect Flag.

Instead, we adopt a lightweight keyed hash to construct HiddenSample as follows.

$$HiddenSample = H(Flag||DataHash||HopCount||K_{i(i+1)}).$$

We include HopCount as the input because its integrity should also be protected. Otherwise, the attacker may tamper with HopCount to breach equal-probability sampling (Section IV-C).

We now analyze the security of our HiddenSample design. First, we append the shared key as a suffix to prevent the length extension attack possible when the key used as a prefix [36], [37]. Second, HiddenSample is robust against hash collision. Through a hash collision attack, the attacker aims to forge some or all of Flag, HopCount, and DataHash without knowing the key $K_{i(i+1)}$, given a specific value of HiddenSample to satisfy. Given an n-bit hash result, a brute-force attack requires $\mathcal{O}(2^n)$ trials to find a pair of inputs with the same hash result while the faster Birthday attack takes $\mathcal{O}(2^{n/2})$ [38]. In comparison with finding any pair of collided inputs, it should be harder to find a specific message $(Flag||DataHash||HopCount||K_{i(i+1)})$ such that its hash result is collided with a given HiddenSample. Consider a 256-bit HiddenSample (Section VI). It requires more than $\mathcal{O}(2^{256/2}) = \mathcal{O}(2^{128})$ trials to forge HiddenSample through a hash collision attack. This is commonly deemed as impractical for an attacker.

We use only one round of computation and comparison of HiddenSample to cover three possible cases (i.e., Flag = 0, Flag = 1, and altered packet). Upon received a packet, a router R_i recovers the hidden sample state by first calculating:

$$HiddenSample' = H(0||DataHash||HopCount||K_{(i-1)i}).$$

If HiddenSample' matches the carried HiddenSample in the received packet, router R_i makes sure that the packet is not sampled by its previous hop and need not verify the ValidationProof field. Otherwise, either the packet is sampled by R_{i-1} (i.e., Flag = 1) or altered (e.g., in fields of Flag, DataHash, or HopCount) in between R_{i-1} and R_i, router R_i proceeds with verifying ValidationProof (Section IV-E).

E. Validation Proof

If a router samples a packet, it computes ValidationProof following existing schemes [39].

$$\begin{aligned}&ValidationProof \\ &= \mathrm{MAC}_{K_{i(i+1)}}(Metadata||HopCount||Concealment),\end{aligned}$$

where Metadata = SessionID||Timestamp||DataHash. We discuss how such a design of ValidationProof can reveal packet mis-forwarding and tampering along with its verification process.

A router verifies ValidationProof if it determines that Flag is not equal to 0 (Section IV-D). To verify the integrity of packet payload, the router first recomputes the hash of the received payload and compares with DataHash. If they do not match, the payload may have been modified and the packet is discarded. Otherwise, the router continues to compute ValidationProof′ as follows:

$$\begin{aligned}&ValidationProof' \\ &= \mathrm{MAC}_{K_{(i-1)i}}(Metadata||HopCount||Concealment).\end{aligned}$$

If ValidationProof′ is equal to ValidationProof, verification succeeds. Otherwise, verification fails and the packet is discarded.

F. Verification

If a router does not sample a packet, it needs to verify the carried ValidationProof if the packet is unsampled by the previous hop. The router first uses the HiddenSample field to verify whether this is so (as in Section IV-D). If the packet is not unsampled in the previous hop (i.e., Flag ! = 0), then the router verifies the ValidationProof field as detailed in Section IV-E.

G. Update

If a router samples a packet, it updates Hummingbird header fields before forwarding the sampled packet as follows (Section IV-D, IV-E):

$$HopCount = HopCount + 1,$$
$$HiddenSample = H(1||DataHash||HopCount||K_{i(i+1)}),$$
$$ValidationProof = \mathrm{MAC}_{K_{i(i+1)}}(Metadata||HopCount||Concealment).$$

If a router does not sample a packet or compute its credentials into ValidationProof (Section IV-E), it should also update various Hummingbird header fields before forwarding the packet to the next hop. This mainly aims to prevent packet correlation and guarantee the hidden sample state. Moreover, if the router decides to verify the packet's ValidationProof (Section IV-F), update follows verification. Specifically, the update process targets the following three fields:

$$HopCount = HopCount + 1,$$
$$HiddenSample = H(0||DataHash||HopCount||K_{i(i+1)}),$$
$$ValidationProof = \textit{128-bit random value}.$$

Of particular emphasis is updating ValidationProof with a random value. We make it appear as random even if it is not computed as for a sampled packet in Section IV-E. If it remains unchanged, the attacker notices this by correlating the packet before and after it traverses the router. An unmodified ValidationProof reveals that the packet is not sampled on the router. This violates our goal of hiding the sample state via HiddenSample. Since ValidationProof of an unsampled packet need not be verified on the next hop, we therefore guarantee its randomness simply using a fast-to-generate random value yet without sacrificing correctness.

H. Hummingbird Algorithm

Finally, we wrap up Hummingbird design with an algorithmic synthesis of the aforementioned functions. Together with Algorithm 1, Algorithm 2 formalizes Hummingbird-enabled router functionalities in Figure 2. `EpSampling` (Algorithm 1), `Construction`, `Verification`, and `Update` (lines 8-31 in Algorithm 2) are detailed in Section IV-C∼Section IV-G, respectively. We now complement them with how the source initiates path validation over a packet (i.e., `SrcInitialization`), how the intermediate trusted routers process the packet (i.e., `Construction`, `Verification`, and `Update`), and how the destination finally accepts the packet (i.e., `DstVerification`).

Algorithm 2: Hummingbird Validation Functions

```
1  Function SrcInitialization:
2      //assume shared key $K_{ij}$ between $R_i$ and $R_j$;
3      $DataHash \leftarrow H(payload)$;
4      $SessionID \leftarrow$ identifier of the current session;
5      $Timestamp \leftarrow$ creation time of packet;
6      $Metadata \leftarrow DataHash||SessionID||Timestamp$;
7      $Concealment \leftarrow \mathrm{MAC}_{K_{0n}}(Metadata)$;
8  Function Construction:
9      $HopCount \leftarrow HopCount + 1$;
10     $HiddenSample$
         $\leftarrow H(1||DataHash||HopCount||K_{i(i+1)})$;
11     $ValidationProof \leftarrow$
         $\mathrm{MAC}_{K_{i(i+1)}}(Metadata||HopCount||Concealment)$;
12 Function Verification:
13     $HiddenSample'$
         $\leftarrow H(0||HopCount||DataHash||K_{(i-1)i})$;
14     if $HiddenSample' \; ! = HiddenSample$ then
15         $DataHash' \leftarrow H(payload)$;
16         if $DataHash' == DataHash$ then
17             $\delta = Metadata||HopCount||Concealment)$;
18             $ValidationProof' \leftarrow \mathrm{MAC}_{K_{(i-1)i}}(\delta)$;
19             if $ValidationProof' == ValidationProof$
                then
20                 $Flag \leftarrow 0$;
21                 Update;
22             else
23                 Discard the packet;
24         else
25             Discard the packet;
26     else
27         Update;
28 Function Update:
29     $HopCount \leftarrow HopCount + 1$;
30     $HiddenSample$
         $\leftarrow H(0||HopCount||DataHash||K_{i(i+1)})$;
31     $ValidationProof \xleftarrow{\$} \{0{,}1\}^{128}$;
32 Function DstVerification:
33     $DataHash' \leftarrow H(payload)$;
34     $Metadata' \leftarrow DataHash'||SessionID||Timestamp$;
35     $Concealment' \leftarrow \mathrm{MAC}_{K_{0n}}(Metadata')$;
36     if $Concealment' == Concealment$ then
37         Accept the packet;
38     else
39         Discard the packet;
```

`SrcInitialization` runs on the source and instantiates packet metadata and source signature. Specifically, packet metadata consists of SessionID, Timestamp, and DataHash (lines 3-6 in Algorithm 2). Then the source signs Metadata using its shared key K_{0n} with the destination with CMAC [40] (line 7):

$$Concealment = \mathrm{MAC}_{K_{0n}}(Metadata).$$

Concealment simultaneously validates packet integrity (DataHash) and freshness (SessionID and Timestamp) as well as source authenticity (K_{0n}).

`Construction`, `Verification`, and `Update` run on intermediate trusted routers. They are responsible for the validation management. Once a router receives a packet, it first invokes `EpSampling` to check whether the packet is sampled. If the returned result shows that the packet is sampled, `Construction` is invoked to generate a new proof using the router's credential. Otherwise, `Verification` starts to verify the carried proof. Once the verification succeeds, the router executes `Update` to update the proof.

`DstVerification` runs on the destination and builds the last defense against packet modification and injection (lines 33-39 in Algorithm 2). It first recomputes DataHash, atop of which Concealment is recomputed. The destination accepts the packet only of the recomputed Concealment matches that carried in the packet.

V. Security

In this section, we analyze Hummingbird security against the Dolev-Yao model [41]. It is a commonly used adversary model by existing path validation solutions [5], [12], [14]. Specifically, the attacker can observe, drop, inject, replay, and forward or alter packets.

A. Proof Unforgeability

A major security goal of path validation should guarantee that the attacker can hardly forge a valid proof. In our Hummingbird design, protected proof fields include HiddenSample, ValidationProof, and Concealment. We use SHA3-256 [42] to compute HiddenSample and CMAC [43] to compute ValidationProof and Concealment. The security of these fields thus depend on the security of SHA3-256 and CMAC.

Theorem 2. *Hummingbird proofs protected by SHA3-256 and CMAC can hardly be forged by a probabilistic polynomial-time-bounded attacker.*

Proof. We have already demonstrated HiddenSample security by robustness of SHA3-256 against hash collision in Section IV-D. We hereby demonstrate security of ValidationProof and Concealment through computation hardness to crack CMAC [44]. Without knowing shared keys among routers, the attacker has to forge proofs via brute-force attacks. The probability of forging a valid 128-bit ValidationProof or Concealment is 2^{-128}, while the probability of forging both is down to 2^{-256}. Take the fastest supercomputer—Fugaku—with a speed of 442 PFLOP/s as example [45]. Even if it can complete one trial per FLOP, it still requires 4.2×10^{51} years to brute-force 2^{256} trials on average. This would be impractically time-consuming for the attacker. □

B. Packet Alteration

The attackers may directly modify packet payloads as they can hardly forge proof fields in the packet header. After modifying the payload of a packet, the attacker needs to recompute DataHash and replace the original one to evade integrity check on intermediate routers. It is possible that a modified packet will not be sampled and verified on intermediate routers again. However, we also compute DataHash into Concealment that must be verified on the destination. By Theorem 2, Concealment with a modified DataHash cannot hardly be forged by the attacker.

C. Packet Injection

The attacker may inject a forged packet or replayed packet to routers. Hummingbird is secure against both types of injection attacks.

- To forge a packet from scratch, the attacker has to forge its corresponding Hummingbird header fields such as HiddenSample, ValidationProof, and Concealment. Without knowing the involved secret key, the attacker can hardly forge these fields by Theorem 2.
- The attacker may also resort to replaying a captured valid packet [46]. Replayed packets can be detected using packet Timestamps [12]. Specifically, Timestamp records the creation time of a packet and thus represents packet freshness. If the time gap between Timestamp and the current time exceeds a threshold, the packet is considered obsolete and discarded. Furthermore, we include Timestamp in the computation of proof fields such as HiddenSample and ValidationProof. The attacker cannot forge HiddenSample and ValidationProof with a tampered Timestamp either.

D. Packet Deviation

When the attacker deviates a packet to a different path than enforced, the deviated can be deemed as either injected or forged. In either case, the deviated packet can be easily detected and discarded (Section V-A$\sim$Section V-C)

VI. Evaluation

In this section, we evaluate the performance of Hummingbird through extensive experiments. The results demonstrate that Hummingbird can guarantee an equal-probability sampling over dynamic paths with unknown length. It can process packets fairly fast and achieve a comparative throughput to the state-of-the-art static path validation solution. The overhead of Hummingbird header fields is 101 bytes irrespective with path length. Hummingbird can achieve a throughput of 1.48 Gbps given a single core and of 8.24 Gbps given four cores with a 10 Gbps link. Such Hummingbird performance is comparative to that of state-of-the-art static path validation solutions.

A. Methodology and Settings

We implement Hummingbird using the Click router [26]. To measure the throughput, we directly connect two machines following the design of OPT [12] and EPIC [14]. Each machine uses an Intel i7-9700 CPU (3.00 GHz) with an Ubuntu 18.04 operating system. We use Intel AES-NI hardware instructions [47] to accelerate cryptography computation. Since Hummingbird is the only path validation solution that supports fully dynamic paths (Table I), we derive a non-sampling baseline out of Hummingbird for efficiency comparison. Specifically, the baseline solution follows the design philosophy in Section IV without including HiddenSample. It thus enables routers to always verify proofs in the received packets and contruct proofs of their own for the next hop to verify.

The evaluation aims to answer the following questions:

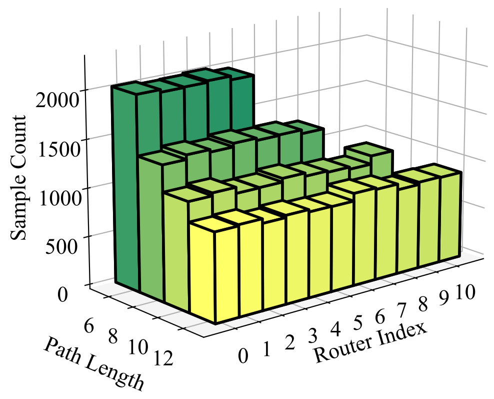

Fig. 4: The number of packets out of 10,000 sampled on a router yet not on its downstream routers with varying path length.

- Can Hummingbird achieve an equal-probability sampling of packets?
- Can Hummingbird leverage probabilistic sampling toward being practically efficient?

B. Sampling Probability

We have proved that Hummingbird can guarantee an equal-probability sampling by Theorem 1. Given P packets over an N-hop path, the number of packets sampled by a router yet not by its downstream routers approximates as $P/(N-1)$. The destination does not participate in sampling. Figure 4 shows the number of such packets out of 10,000 test packets with varying path length. The key observation is that for any path length, sample counts are uniformly distributed across different hops. Consider the 6-hop path for example. The expected sample count in theory should be $10000/(6-1) = 2000$. The collected statistics for sample counts are $\{2020, 1993, 1998, 2014, 1975\}$, well matching with the theoretical distribution.

C. Proof Size

We measure the communication overhead of Hummingbird in terms of proof size. We expect fields to be as concise as possible so that packets can carry more payloads. Table III shows the size of each Hummingbird field detailed in Section IV-B under a 128-bit security level. As with Baseline, Hummingbird guarantees a constant proof size regardless of payload size and path length. We include the PPV as state-of-the-art constant-size–proof design as well. Hummingbird has a proof size of 101 bytes, in comparison with 69 bytes for Baseline and 64 bytes for PPV. The major size difference arises from the additional HiddenSample field that Hummingbird introduces to hide the sample state. Albeit Hummingbird needs a relatively longer proof than Baseline does, it outperforms Baseline in processing delay, throughput, and the ultimate goodput. We next report corresponding comparison results.

D. Processing Speed per Function

Packet processing on a Hummingbird router involves six functions—`SrcInitialization`, `EpSampling`, `Construction`, `Verification`, `Update`, and

TABLE III: Proof size (in byte) comparison under a 128-bit security level.

Field	Solution		
	Hummingbird	Baseline	PPV
HiddenSample	32	/	/
HopCount	1	1	/
DataHash	16	16	/
SessionID	16	16	16
Timestamp	4	4	16 (FlowID)
Concealment	16	16	16 (MVF)
ValidationProof	16	16	32 (Others)
Total	101	69	64

`DstVerification` (Section IV). `SrcInitialization` and `DstVerification` run on only the source and the destination, respectively. `EpSampling` runs on the source and intermediate routers. If they `EpSampling` returns 1, they further invoke `Construction` for generating validation proofs. Otherwise, they invoke `Update` for randomizing the validation proof and hiding the sample state. `Verification` runs on the intermediate and destination routers for verifying Hummingbird header fields. `Verification` accordingly splits into two sub-functions. One uses HiddenSample to verify whether a packet is sampled by the previous hop (line 13 in Algorithm 2). If yes, the other uses ValidationProof to verify the proof of the previous hop (lines 14-27 in Algorithm 2).

We measure the average processing speed of each function using 100,000 packets each with 1,000-byte payload. Table IV reports the measurement results in comparison with Baseline. For `SrcInitialization`, `Update`, `Verification` of ValidationProof, and `DstVerification` that have similar functionalities in Baseline, Hummingbird yields a faster processing. In contrast, `EpSampling`, `Construction`, and `Verification` of HiddenSample exist only in Hummingbird, with no counterpart in Baseline. `EpSampling` and `Verification` of HiddenSample are fairly fast, taking $0.26 + 1.68 = 1.94 \times 10^{-7}$ seconds. `Construction` takes 5.61×10^{-7} seconds. Although `Construction` is a bit slower, it takes effect only when `EpSampling` returns Flag = 1 with a low probability. We do not showcase the measurements with varying path length and payload size as packet processing time is insensitive to them (Section VI-C).

E. Processing Speed per Router

Figure 5 reports packet processing time across different routers. Again, the statistics are collected over 100,000 packets each with 1,000-byte payload.

Source router. The packet processing time of Hummingbird on the source can be estimated as:

$$T_{src} = T_{init} + T_{constr}.$$

T_{init} and T_{constr} denote the time of `SrcInitialization` and `Construction`, respectively. Under a 128-bit security level, Hummingbird needs 1.44 μs to construct proofs. In comparison, Baseline takes 1.15 μs. The slightly extra time Hummingbird takes is to compute HiddenSample.

Intermediate router. Let P_n denote the probability of a packet being sampled on the nth router. Let T_s, T_c, T_{vh}, T_{vv},

TABLE IV: Packet processing speed (10^{-7} second) per function using 100,000 test packets each with 1,000-byte payload.

Solution	Packet Processing Function						
	SrcInitialization	EpSampling	Construction	Verification		Update	DstVerification
				HiddenSample	ValidationProof		
Baseline	11.48	/	/	/	8.84	2.86	2.67
Hummingbird	8.74	0.26	5.61	1.68	8.70	1.68	2.74

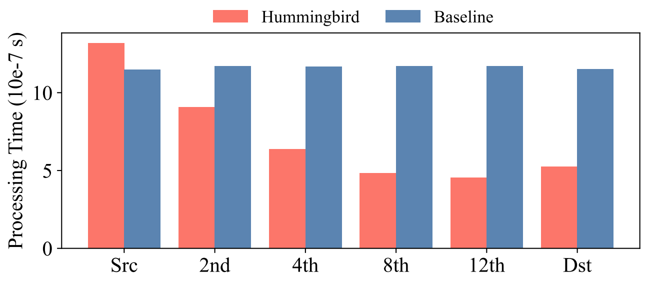

Fig. 5: Packet processing speed (10^{-7} second) per router using 100,000 test packets each with 1,000-byte payload.

and T_u denote the time of `EpSampling`, `Construction`, `Verification` of HiddenSample, `Verification` of ValidationProof, and `Update`, respectively. The overall packet process time of Hummingbird on an intermediate router can be defined as:

$$\begin{aligned} T_{ir} &= T_s + P_n T_c + (1 - P_n)(T_{vh} + P_{n-1} T_{vv} + T_u) \\ &= T_s + \frac{1}{n}(T_c + T_{vv}) + \frac{n-1}{n}(T_{vh} + T_u). \end{aligned}$$

Different from the packet processing time on the source, the packet processing time on an intermediate router is no longer a constant. It depends on the sampling probability, which decreases with HopCount (Algorithm 1). The more routers that the packet has traversed, the less likely it is sampled on the current router. The less process time it thus may experience as well. Furthermore, this also indicates a shorter process time on the next hop where ValidationProof is less likely to be necessary for `Verification`. As shown in Figure 5, the average packet processing time of Hummingbird on the second hop is 0.91 μs, while it decreases to 0.50 μs on the eighth hop. In contrast, Baseline imposes a constant processing time about $0.88 + 0.29 = 1.17$ μs on intermediate routers, where verifying and updating proofs contribute 0.88 μs and 0.29 μs, respectively. The efficiency gap between Hummingbird and Baseline thus increases with path length. Note that the processing time of Hummingbird on the source and a specific intermediate router is insensitive to path length. This further demonstrates that Hummingbird supports fully dynamic paths with unknown path length.

Destination router. The packet processing time of Hummingbird on the destination can be estimated as:

$$\begin{aligned} T_{dst} &= T_{vh} + P_{n-1} T_{vv} + T_{dv} \\ &= T_{vh} + \frac{1}{n-1} T_{vv} + T_{dv}, \end{aligned}$$

where P_{n-1} is the probability that the packet is sampled by the previous router; T_{vh}, T_{vv}, and T_{dv} denote the time of `Verification` of HiddenSample, `Verification` of ValidationProof, and `DstVerification`, respectively. As path length increases, the probability of the second-last hop sampling a packet decreases. Therefore, the destination is more likely to take less time for `DstVerification`. For example, when the path length is 8, the destination takes 0.57 μs to process the packet. When the path length scales to 12, the destination only takes 0.52 μs. In contrast, Baseline imposes a slower processing on the destination, being about 1.15 μs irrespective with path length.

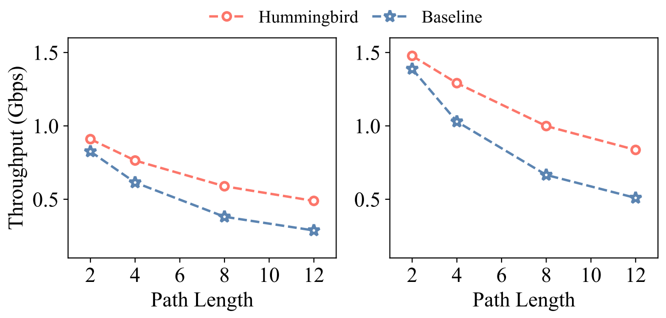

Fig. 6: Comparison of throughput with varying payload size and path length.

F. Throughput

We now measure the throughput of Hummingbird in comparison with that of Baseline. As shown in Figure 6, throughput of both Hummingbird and Baseline increases with payload and decreases with path length. First, the payload size does not affect the processing time (Section VI-D). A larger payload size yields more data being processed and transmitted per unit of time. Thus a higher throughput is achieved. For example, when the payload size increases from 500 bytes to 1000 bytes, the throughput of Hummingbird increases from 0.91 Gbps to 1.48 Gbps given a 2-hop path. Second, the total processing time of a packet along an entire path increases as the path gets longer. It thus takes more time to transmit a fixed-size packet along a longer path; this decreases throughput. For example, given a 1000-byte payload, the throughput of Hummingbird decreases from 1.48 Gbps to 1.00 Gbps when path length increases from 2 to 8.

Hummingbird outperforms Baseline in terms of a much higher throughput as path length increases. Consider the 1000-byte payload scenario for example. When path length is 8, Hummingbird's throughput is 49.3% higher than that of Baseline. The improvement increases to 64.2% when path length reaches 12.

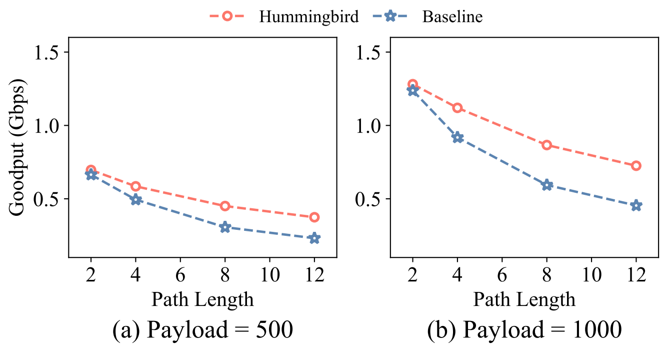

Fig. 7: Comparison of goodput with varying payload size and path length.

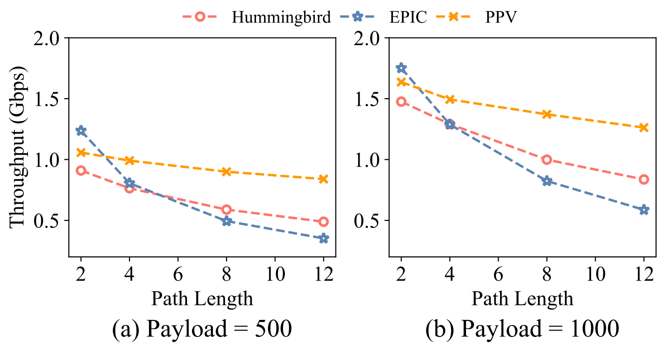

Fig. 8: Comparison of throughput with varying payload size and path length.

G. Goodput

We report the measurements of goodput that reflects the amount of effective data (i.e., payload) transmitted per unit time. Let the goodput ratio define the ratio of payload size and packet size. Then the goodput can be quantified as [12], [14]:

$$\text{goodput} = \text{throughput} \times \text{goodput ratio}.$$

As reported in Table III, the proof size of Hummingbird and Baseline is 101 bytes and 69 bytes, respectively. Given a fixed proof size, the goodput ratio increases with packet size. For example, when the payload size is 500 bytes, the goodput ratio of Hummingbird and Baseline is 76.57% and 80.52%, respectively. When the payload size increases to 1000 bytes, the goodput ratio of Hummingbird and Baseline increases to 86.73% and 89.21%, respectively. We then measure the goodput using the goodput ratio and the throughput reported in Figure 6. As shown in Figure 7, Hummingbird yields a higher goodput than Baseline does. The benefit increases with path length.

H. Comparison with Static Path Validation

To justify the scale of throughput and goodput reported in Figure 6 and Figure 7, we implement the state-of-the-art static path validation solutions—EPIC [14] and PPV [13]—as a reference. Both of them require the entire forwarding path to be known as a priori. EPIC enforces per-hop validation while PPV, like Hummingbird, samples packets to validate for boosting efficiency. As shown in Figure 8,

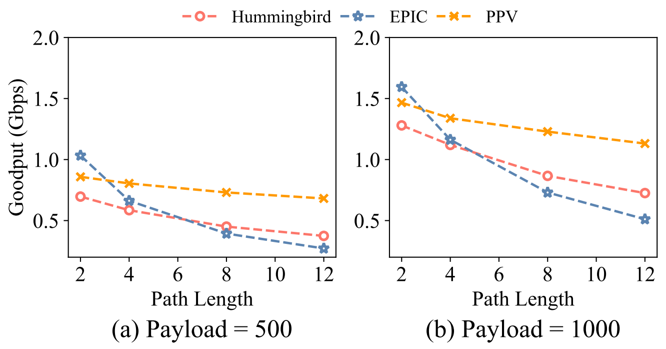

Fig. 9: Comparison of goodput with varying payload size and path length.

EPIC outperforms PPV and Hummingbird given relatively short forwarding paths. For example, given 1,000-byte packets along a 2-hop path, EPIC yields a 1.71 Gbps[1] throughput while Hummingbird throughput amounts to 1.48 Gbps. This is because Hummingbird performs more complex calculation to deal with dynamic paths. However, as paths become longer, Hummingbird starts outperforming EPIC. Consider still 1,000-byte packets in use for example. When path length increases to 12, EPIC throughput is 0.59 Gbps while Hummingbird yields a 42.37% higher throughput of 0.84 Gbps. This is because that the longer the forwarding path is, the more chances for Hummingbird to reduce the number of validations by equal-probability sampling. With less validation overhead comes a higher throughput than that of EPIC. The preceding observations apply also to the goodput comparison reported in Figure 9. Different from the comparison between EPIC and Hummingbird, PPV constantly yields a higher throughput and goodput than Hummingbird does. This is because PPV does not enforce some of Hummingbird's computations to support dynamic path validation as well hidden sampling status.

I. Tradeoff between Security and Efficiency

As Hummingbird adopts probabilistic validation toward tradeoff between security and efficiency, we investigate how sampling impacts security and efficiency. The evaluation is still compared against PPV [13] and EPIC [14].

Security. Hummingbird as well as PPV do not enforce hop-wise validation as EPIC does. The forwarding misbehavior of a certain path segment may be left undetected if they sample no packets right when packets traverse the segment. However, a flow usually consists of a large number of packets. Jointly using validation results of co-flow packets, we can still detect forwarding misbehaviors [13].

The key interest of security evaluation is how many packets needed to cover the entire path [13]. EPIC performs hop-wise validation so that one packet is sufficient, regardless of the path length. Given an N-hop path, PPV needs N packets to achieve the same effect [13]. This is because PPV samples each packet only once and after being sampled, a packet carries only one

[1]Note that the 1.71 Gbps throughput of EPIC is relatively lower than the original result in [14]. We suspect that this is because the test server used in [14] can support up to 40 Gbps bandwidth while our server support up to 10 Gbps.

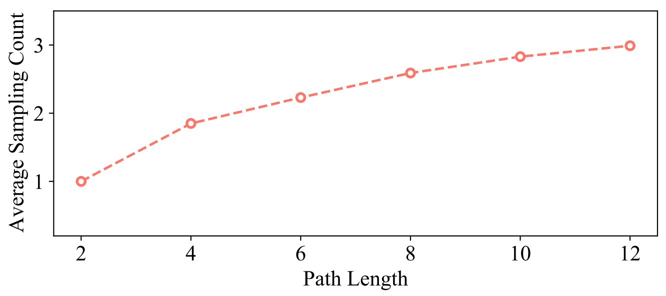

Fig. 10: Average sampling count by Hummingbird throughput packet forwarding with varying path length.

TABLE V: Comparison of packet count needed to detect forwarding misbehaviors across the entire path.

Solution	Path Length			
	$n = 2$	$n = 4$	$n = 8$	$n = 12$
EPIC	1	1	1	1
Hummingbird	1	2	3	4
PPV	1	4	8	12

segment's proof. The re-sampling technique enables Hummingbird to likely sample a packet more than once. Security of Hummingbird is thus between that of PPV and EPIC. Since the sampling rate of a packet is $1/k$ in Hummingbird, where k is the hop-index of routers (Section IV-C), the expectation of a packet being sampled x times per transmission is equal to $E(x) = 1 + \frac{1}{2} + ... + \frac{1}{k} + ... + \frac{1}{N} \approx \ln(N) + 0.5772$. Figure 10 shows the average sampling count of a packet out of 10,000 test packets with varying path length. Consider an 8-hop path for example. The expected sampling count is $\ln(8) + 0.5772 = 2.66$. The measurement result is $25856/10000 = 2.59$, well conforming to the theoretical distribution. Considering that one sampling corresponds to the distance of one hop, the average distance a packet can cover is also $\ln(N) + 0.5772$. Thus, the number of packets that Hummingbird needs to cover the entire path is $N/(\ln(N) + 0.5772)$. Table V summarizes the number of packets that different solutions needed to cover the entire path. Hummingbird stays in between PPV and EPIC in terms of security.

Efficiency We further evaluate how the tradeoff of security by Hummingbird and PPV helps them improve efficiency. Performance metrics of interest are the total end-to-end processing time across the entire forwarding path and the average processing time on every enroute node. As shown in Figure 11 and Figure 12, as path length scales, Hummingbird is faster than EPIC and slower than PPV. When path length is 12, the average processing time on each router is 0.86 μs for Hummingbird and 1.14 μs for EPIC.

J. Multi-core Acceleration

Finally, we investigate the effect of multi-core acceleration for Hummingbird. We implement a multi-core version of Hummingbird using DPDK [27] on an Intel Xeon E5-2630 v3 server [48]. The server features with 8 cores, 16 GB memory, and two hardware network-interface cards (NICs). Such NICs associate with Intel Corporation I350 Gigabit Network Connection and Intel Corporation Ethernet Controller

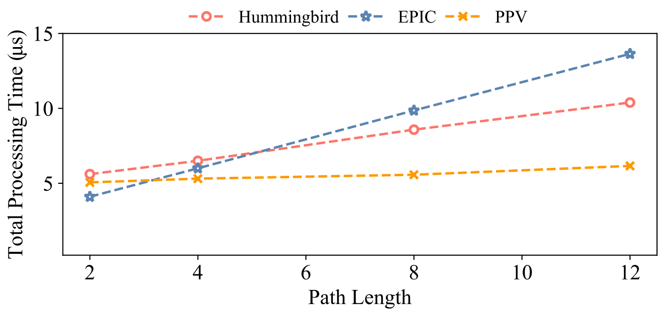

Fig. 11: Comparison of total end-to-end processing time with varying path length.

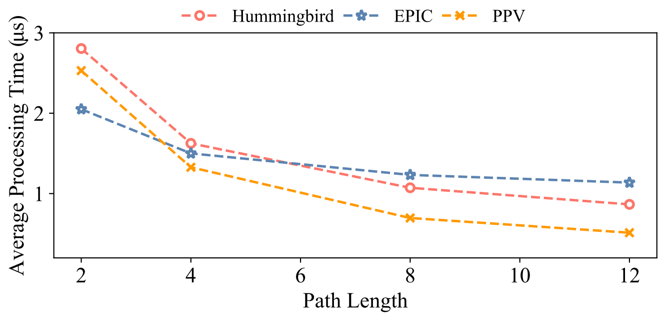

Fig. 12: Comparison of average processing time on each node with varying path length.

X710, offering a 10 Gbps link. We expect throughput to increase with core count available for packet processing. Memory size for temporary packet storage, however, imposes a practical restriction. Even though the router has sufficient on-core computation resources to process more packets, it can hardly do so with little remaining memory space for storing those packets. Due to the relatively limited 16 GB memory, the server we use reaches such peak usage when 4 cores are allocated for packet processing. Figure 13 reports Hummingbird throughput with varying core count and payload size on a 4-hop path. First, it echoes with preceding results that given a fixed number of cores, throughput increases with payload size. Second, of more focus of multi-core evaluation is that given fixed payload size, throughput increases with the number of allocated cores. Validating 1,000-byte packets, Hummingbird achieves a 2.08 Gbps throughput using only one core. It can be accelerated to 4.27 Gbps and 8.24 Gbps given up to 2 cores and 4 cores, respectively.

VII. Conclusion

We have proposed Hummingbird as the first path validation solution that supports fully dynamic paths. In contrast with existing path validation solutions, Hummingbird can better suit for networks with flexible routing policies. It features an equal-probability sampling technique to enforce efficient validation of packets albeit with unknown path length. The sample state is also hidden via a lightweight, non-cryptographic scheme. Both strategies prevent attackers from distinguishing sampled and unsampled packets and thus prevent them from selectively attacking packet forwarding. We validate security and efficiency of Hummingbird through both theoretical proof and

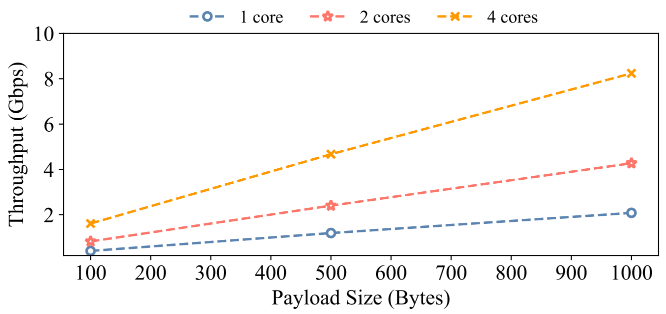

Fig. 13: Hummingbird throughput with varying core count and payload size on a 4-hop path.

experimental evaluation. The results demonstrate that Hummingbird can guarantee an equal-probability sampling over dynamic paths with unknown length. It can process packets fairly fast and achieve a comparative throughput to the state-of-the-art static path validation solution.

We suggest a lightweight enhancement for Hummingbird to support path re-construction on the destination. Since Hummingbird aims to validate dynamic paths, it is infeasible to use a constant packet field to record forwarding paths as in static path validation [12], [14], [15]. It is also prohibitively inefficient to simply record all the possible dynamic paths in the packet header [19]. Inspired by the hierarchy division technique from Atlas [19], we introduce a dynamically updated field, RTag, to record a divergence node (i.e., a router that has multiple successors) that a packet traverses. RTag can be appended to the Concealment field shown in Figure 3 so that routers can process it without intervening the path validation procedure as described in Section IV. The destination leverages RTags in received packets for path re-construction as follows.

- While the source and destination are designated with a set of dynamic paths within a session, we first randomly bound SessionID to a fixed static path in the set. The bounding choice should be known to both the source and destination as well as the enroute trusted routers.
- Once a trusted router receives a packet, it needs to add no RTag if it is on the path bounded with SessionID. Otherwise, the router adds an RTag into the header to record its identifier. The first RTag indicates that the packet has already deviated from the original path bounded with SessionID. All subsequent trusted routers need to append their RTags afterwards.
- When the destination receives the packet, it can reconstruct the path of trusted routers by splicing 1) trusted routers on the SessionID bounded path prior to the first RTag's indicated router and 2) trusted routers corresponding to all the RTags.

For future work, we plan to unleash Hummingbird speed through hardware acceleration. The reported results in Section VI are measured over software routers. Albeit software routers tend to be more and more prevalent in large-scale data centers, hardware routers are still critical for building a high-speed infrastructure. We plan to offload as many Hummingbird functionalities to FPGA boards [5] or Smart NICs [49]–[53] as possible.

Acknowledgment

We would like to sincerely thank the Editors and Reviewers of IEEE Transactions on Information Forensics and Security for your review efforts and helpful feedback. We also wish you health and safety during the pandemic.

The work is supported in part by National Natural Science Foundation of China under Grant No. 62172358, National Key R&D Program of China under Grant No. 2020AAA0107705, National Natural Science Foundation of under Grant No. 62032021, Zhejiang Key R&D Plan under Grant No. 2019C03133, Leading Innovative and Entrepreneur Team Introduction Program of Zhejiang under Grant No. 2018R01005, Alibaba-Zhejiang University Joint Institute of Frontier Technologies, and Research Institute of Cyberspace Governance in Zhejiang University.

References

[1] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "Scion: Scalability, control, and isolation on next-generation networks," in *S&P*. IEEE, 2011, pp. 212–227.
[2] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk, and P. Szalachowski, "The scion internet architecture," *Communications of the ACM*, 2017.
[3] K. Bu, A. Laird, Y. Yang, L. Cheng, J. Luo, Y. Li, and K. Ren, "Unveiling the mystery of internet packet forwarding: A survey of network path validation," *ACM Computing Surveys*, vol. 53, no. 5, pp. 1–34, 2020.
[4] SCIONLab. [Online]. Available: https://www.scionlab.org/
[5] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra, "Verifying and enforcing network paths with icing," in *CoNEXT*, 2011.
[6] H. Birge-Lee, Y. Sun, A. Edmundson, J. Rexford, and P. Mittal, "Bamboozling certificate authorities with {BGP}," in *USENIX Security*, 2018, pp. 833–849.
[7] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "Raptor: Routing attacks on privacy in tor," in *USENIX Security*, 2015, pp. 271–286.
[8] L. D. Amini, A. Shaikh, and H. G. Schulzrinne, "Issues with inferring internet topological attributes," in *Internet Performance and Control of Network Systems III*, vol. 4865. International Society for Optics and Photonics, 2002, pp. 80–90.
[9] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding traceroute anomalies with paris traceroute," in *SIGCOMM*, 2006, pp. 153–158.
[10] X. Yang, D. Clark, and A. W. Berger, "Nira: a new inter-domain routing architecture," *IEEE/ACM transactions on networking*, vol. 15, no. 4, pp. 775–788, 2007.
[11] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. J. Freedman, A. Haeberlen, Z. G. Ives, A. Krishnamurthy *et al.*, "The nebula future internet architecture," in *The Future Internet Assembly*. Springer, 2013, pp. 16–26.
[12] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *SIGCOMM*, vol. 44, no. 4, 2014, pp. 271–282.
[13] B. Wu, K. Xu, Q. Li, Z. Liu, Y.-C. Hu, M. J. Reed, M. Shenk, and F. Yang, "Enabling efficient source and path verification via probabilistic packet marking," in *IWQoS*, 2018.
[14] M. Legner, T. Klenze, M. Wyss, C. Sprenger, and A. Perrig, "Epic: Every packet is checked in the data plane of a path-aware internet," in *USENIX Security Symposium*, 2020, pp. 541–558.
[15] A. He, K. Bu, Y. Li, E. Chida, Q. Gu, and K. Ren, "Atomos: Constant-size path validation proof," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3832–3847, 2020.
[16] F. Yang, K. Xu, Q. Li, R. Lu, B. Wu, T. Zhang, Y. Zhao, and M. Shen, "I know if the journey changes: Flexible source and path validation," in *IWQoS*, 2020, pp. 1–6.
[17] H. Cai and T. Wolf, "Source authentication and path validation with orthogonal network capabilities," in *INFOCOM WKSHPS*, 2015.
[18] ——, "Source authentication and path validation in networks using orthogonal sequences," in *IEEE ICCCN*, 2016, pp. 1–10.

[19] L. Ma, K. Bu, N. Wu, T. Luo, and K. Ren, "Atlas: A first step toward multipath validation," *Computer Networks*, vol. 173, p. 107224, 2020.

[20] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin, "The impact of routing policy on internet paths," in *INFOCOM*, vol. 2. IEEE, 2001, pp. 736–742.

[21] K. M. Carley, *Dynamic network analysis*. na, 2003.

[22] S. Gao and I. Chabini, "Optimal routing policy problems in stochastic time-dependent networks," *Transportation Research Part B: Methodological*, vol. 40, no. 2, pp. 93–122, 2006.

[23] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Contra: A programmable system for performance-aware routing," in *NSDI*, 2020, pp. 701–721.

[24] A. Abhashkumar, K. Subramanian, A. Andreyev, H. Kim, N. K. Salem, J. Yang, P. Lapukhov, A. Akella, and H. Zeng, "Running bgp in data centers at scale." in *NSDI*, 2021, pp. 65–81.

[25] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software*, vol. 11, no. 1, pp. 37–57, 1985.

[26] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[27] DPDK: Data Plane Development Kit,. [Online]. Available: http://dpdk.org/

[28] B. Sengupta, Y. Li, K. Bu, and R. H. Deng, "Privacy-preserving network path validation," *ACM Transactions on Internet Technology*, vol. 20, no. 1, pp. 1–27, 2020.

[29] P. A. Porras, S. Cheung, M. W. Fong, K. Skinner, and V. Yegneswaran, "Securing the software defined network control layer." in *NDSS*, 2015.

[30] Q. Li, X. Zou, Q. Huang, J. Zheng, and P. P. Lee, "Dynamic packet forwarding verification in sdn," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 6, pp. 915–929, 2018.

[31] Q. Li, Y. Chen, P. P. Lee, M. Xu, and K. Ren, "Security policy violations in sdn data plane," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1715–1727, 2018.

[32] P. Zhang, H. Wu, D. Zhang, and Q. Li, "Verifying rule enforcement in software defined networks with rev," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 917–929, 2020.

[33] P. Zhang, F. Zhang, S. Xu, Z. Yang, H. Li, Q. Li, H. Wang, C. Shen, and C. Hu, "Network-wide forwarding anomaly detection and localization in software defined networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 332–345, 2020.

[34] Q. Li, Y. Liu, Z. Liu, P. Zhang, and C. Pang, "Efficient forwarding anomaly detection in software-defined networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2676–2690, 2021.

[35] S. Xi, K. Bu, W. Mao, X. Zhang, K. Ren, and X. Ren, "Ruleout forwarding anomalies for sdn," *IEEE/ACM Transactions on Networking*, 2022.

[36] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *CRYPTO*. Springer, 1996, pp. 1–15.

[37] D. Gligoroski, "Length extension attack on narrow-pipe sha-3 candidates," in *ICT Innovations*. Springer, 2010, pp. 5–10.

[38] M. Bellare and T. Kohno, "Hash function balance and its impact on birthday attacks," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2004, pp. 401–418.

[39] G. Liu, H. Sadok, A. Kohlbrenner, B. Parno, V. Sekar, and J. Sherry, "Don't yank my chain: Auditable nf service chaining," in *NSDI*, 2021, pp. 155–173.

[40] J. Song, R. Poovendran, J. Lee, and T. Iwata, "The aes-cmac algorithm," RFC 4493, June, Tech. Rep., 2006.

[41] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[42] M. J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015.

[43] I. 9797, "Data cryptographic techniques: Data integrity mechanism using a cryptographic check function employing a block cipher algorithm," 1989.

[44] M. Bellare, J. Kilian, and P. Rogaway, "The security of the cipher block chaining message authentication code," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 362–399, 2000.

[45] J. Dongarra, "Report on the fujitsu fugaku system," *University of Tennessee-Knoxville Innovative Computing Laboratory, Technical Report. ICLUT*, 2020.

[46] T. Lee, C. Pappas, A. Perrig, V. Gligor, and Y.-C. Hu, "The case for in-network replay suppression," in *AsiaCCS*, 2017, pp. 862–873.

[47] J. Rott, "Intel advanced encryption standard instructions (aes-ni)," *Technical Report, Intel*, 2010.

[48] Intel Xeon Processor E5-2630 v3,. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/83356/intel-xeon-processor-e52630-v3-20m-cache-2-40-ghz/specifications.html

[49] J. Min, M. Liu, T. Chugh, C. Zhao, A. Wei, I. H. Doh, and A. Krishnamurthy, "Gimbal: enabling multi-tenant storage disaggregation on smartnic jbofs," in *SIGCOMM*, 2021, pp. 106–122.

[50] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren, "Smartnic performance isolation with fairnic: Programmable networking for the cloud," in *SIGCOMM*, 2020, pp. 681–693.

[51] M. Tork, L. Maudlej, and M. Silberstein, "Lynx: A smartnic-driven accelerator-centric architecture for network servers," in *ASPLOS*, 2020, pp. 117–131.

[52] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "Panic: A high-performance programmable nic for multi-tenant networks," in *OSDI*, 2020, pp. 243–259.

[53] S. Pirelli and G. Candea, "A simpler and faster nic driver model for network functions," in *OSDI*, 2020, pp. 225–241.

Anxiao He received the B.Sc. degree in computer science from the College of Computer Science and Technology, Zhejiang University, in 2020. He is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University. His research interest includes network security.

Xiang Li received the B.Sc. degree in computer science from the College of Computer Science and Technology, Zhejiang University, in 2021. She is currently pursuing the Ph.D. degree with the College of Cyberspace Science, Harbin Institute of Technology. Her research interest includes privacy protection and federated learning.

Jiandong Fu received the B.Sc. degree in computer science from the College of Computer Science, Hangzhou Dianzi University, in 2019. He is currently pursuing the master degree with the College of Computer Science and Technology, Zhejiang University. His research interest includes network security.

Haoyu Hu received the B.Sc. degree in computer science from the College of Computer Science and Technology, Zhejiang University, in 2022. His research interests include network security and computer vision.

Kai Bu received the B.Sc. and M.Sc. degrees in computer science from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2006 and 2009, respectively, and the Ph.D. degree in computer science from The Hong Kong Polytechnic University, Hong Kong, in 2013. He is currently an Associate Professor with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research interests include network security and computer architecture. He is a member of the ACM, the IEEE, and the CCF. He is a recipient of the Best Paper Award of IEEE/IFIP EUC 2011 and the Best Paper Nominee of IEEE ICDCS 2016.

Chenlu Miao received the B.Sc. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2021. She is currently pursuing the master degree in computer science and technology with Zhejiang University, Hangzhou, China. Her current research interest is computer architecture.

Kui Ren is Professor and Associate Dean of College of Computer Science and Technology at Zhejiang University, where he also directs the Institute of Cyber Science and Technology. Before that, he was SUNY Empire Innovation Professor at State University of New York at Buffalo. Kui's research interests include Data Security, IoT Security, AI Security, and Privacy. He received many recognitions including Guohua Distinguished Scholar Award of ZJU, IEEE CISTC Technical Recognition Award, SUNY Chancellor's Research Excellence Award, Sigma Xi Research Excellence Award, NSF CAREER Award, etc. Kui has published extensively in peer-reviewed journals and conferences and received the Test-of-time Paper Award from IEEE INFOCOM and many Best Paper Awards from IEEE and ACM, including ACM MobiSys, IEEE ICDCS, IEEE ICNP, IEEE Globecom, ACM/IEEE IWQoS, etc. Kui is a Fellow of ACM and IEEE. And he serves on the editorial boards of many IEEE and ACM journals. He also serves as Chair of SIGSAC of ACM China Council, a member of ACM ASIACCS steering committee, and a member of ST Committee of Ministry of Education of China.