Prelude: Ensuring Inter-Domain Loop-Freedom in SDN-Enabled Networks

Arnaud Dethise
KAUST

Marco Chiesa
KTH Royal Institute of Technology

Marco Canini
KAUST

ABSTRACT

Software-Defined eXchanges (SDXes) promise to improve the inter-domain routing ecosystem through SDN deployment. Yet, the naïve deployment of SDN on the Internet raises concerns about the correctness of the inter-domain data-plane. By allowing operators to deflect traffic from default BGP routes, SDN policies can create permanent forwarding loops that are not visible to the control-plane.

We propose PRELUDE, a system for detecting SDN-induced forwarding loops between SDXes with high accuracy without leaking private routing information of network operators. To achieve this, we leverage Secure Multi-Party Computation (SMPC) techniques to build a novel and general privacy-preserving primitive that detects whether any subset of SDN rules might affect the same portion of traffic without learning anything about those rules. We then leverage this primitive as the main building block of a distributed system tailored to detect forwarding loops among any set of SDXes. We leverage the particular nature of SDXes to further improve the efficiency of our SMPC solution.

The number of valid SDN rules rejected by our solution is 100x lower than previous privacy-preserving solutions, and provides better privacy guarantees. Furthermore, our solution naturally provides network operators with some insights on the cost of the deflected paths.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; **Network security**; *Cryptography*; • **Networks** → **Routing protocols**; *Network privacy and anonymity*;

1 INTRODUCTION

In recent years, SDN has transformed the way network operators design and manage networks by supporting highly flexible and fine-grained network control [15]. Yet, deploying SDN in practice is notoriously difficult [18] as the co-existence with legacy devices and routing protocols often leads to unpredictable behaviors.

One of the most notable examples of this phenomenon has been observed by Birkner et al. [4] in the emerging context of SDN-enabled Internet eXchange Points, called Software-Defined eXchanges (SDXes) [13]. SDXes have been deployed in production [2] and are currently being tested by the largest IXPs worldwide. Yet, naïve deployment of such architectures comes with severe potential Internet-wide performance degradation as the growing number of SDXes may increase the chances of creating inter-domain forwarding loops.

These unwanted forwarding behaviors stem from a simple fact: there exists a mismatch between the routing expressiveness of SDN and BGP, the de-facto standard inter-domain routing protocol. SDN at SDXes can be used to create *deflections*, i.e., forwarding decisions that are explicitly inconsistent with the BGP forwarding decisions. Although these have the purpose to optimize traffic delivery, naïvely using them can ultimately prevent BGP from guaranteeing loop-freedom. Given the practical impossibility of replacing BGP in the short term with a clean-slate routing protocol, SDXes must be armed with tools for avoiding forwarding loops and guaranteeing a *safe* (i.e., loop-free) forwarding behavior.

Challenges. Preserving the safety of Internet routing is a fundamental yet complex problem that entails *efficiently* detecting forwarding loops while *preserving the privacy* of routing policies and any other information about the economic relationships among the networks that is deemed confidential [1, 7] (see survey in [6] for details).

Our approach. We devise PRELUDE, a novel approach to efficiently detecting forwarding loops that guarantees a safe forwarding behavior across the Internet, and both higher levels of privacy and lower rejection rate for safe SDN policies (deflections) than state-of-the-art solutions.

PRELUDE is built upon a simple yet powerful network verification primitive, called DISTINCT-MATCH, that allows any two networks to verify whether their SDN rules overlap, i.e., the set of packets matched by the conjunction of these rules is non-empty, without leaking any information about the SDN rules. We leverage recent advancements on Secure Multi-Party Computation (SMPC) to evaluate functions in a privacy-preserving way. To maximize efficiency, PRELUDE performs as little computation as possible within the SMPC machinery.

Our contributions. The main contributions are:

- We analyze the impact of BGP deflections on Internet routing and prove that BGP deflections compliant with traditional yet incomplete customer-provider relationships are loop-free.
- We design DISTINCT-MATCH, a general privacy-preserving primitive that allows any two networks to verify whether their SDN policies overlap.
- We devise and evaluate PRELUDE, a control-plane that runs at the SDXes in parallel with BGP and detects forwarding loops without leaking private routing policies of the SDX members. We prove PRELUDE identifies all forwarding loops involving any two networks connected at two distinct SDXes with *zero* false positives[1] and show that it reduces the number of false positives with more than two SDXes over previous solutions.

Related work. The work closest to ours is SIDR [4], which first observed the problem of SDN-induced forwarding loops and discussed how to detect them while preserving privacy at the cost of accepting high false positive rates. Our solution takes inspiration from other applications of SMPC for IXP Route Server design, such as SIXPACK [7], and the Internet-wide Route Server [1], ultimately leading to almost zero false positives in detecting forwarding loops. We do not rely on the SGX technology as it constraints network operators to rely on very specific hardware and its different and debated level of security, discussed in more details in [7]. We however consider SGX complementary to our solution. Finally, VeriFlow [14] performs verification of SDN rules in single-domain networks that do not require privacy.

2 ON IXPS AND DEFLECTIONS

In this section, we present a background on Internet eXchange Points (IXPs), followed by a discussion on the problems that arise from leveraging the expressiveness of SDN rules to deflect traffic away from BGP paths, a problem that was introduced in [4]. We then prove that there exists a practically relevant class of deflections that do not cause any forwarding loops.

Background on IXPs. IXPs are the high-speed interconnection network infrastructures through which a multitude of Autonomous Systems (ASes), called the IXP *members*, physically interconnect to exchange traffic. Recently, IXPs experienced a sharp growth in customer base and higher volumes of traffic due to reduced path latencies and lower transit costs, ultimately resulting in the existence of over 800 IXPs around the world. At the largest IXPs, hundreds of members exchange ever-increasing volumes of traffic (in the order of 5 Tbps) towards hundreds of thousands of IP prefix destinations.

Routing information at IXPs is exchanged through BGP in the form of path routes towards IP destination prefixes. Network operators specify how to rank, export, and select the routes to be used for forwarding traffic through *BGP policies*. To prevent persistent *forwarding loops* (i.e., packets that are sent in a network cycle), BGP uses a loop-avoidance mechanism. For the scope of this paper, it is worth knowing that this mechanism works as long as *(i)* a single route is used to forward traffic and *(ii)* only this route is exported to other networks.[2]

IXPs are deemed as a hot spot for innovation [8, 13]. Some of the largest ones are today experimenting new SDX architectures. SDXes aim at supporting advanced use cases for both IXP members and IXP operators [8, 9] without requiring any member to deploy SDN switches. Members steer their incoming/outgoing traffic at the SDX by configuring their *SDN policies*, which match flows of traffic (based on Layer 2 to Layer 4 fields) and apply specific actions (such as forwarding a packet to a certain member) [13].

BGP deflections at SDXes. When IXP members install SDN policies at an SDX, the BGP loop-avoidance assumptions may be violated, resulting in forwarding loops [4]. In fact, to support advanced peering applications that need to steer traffic through different members according to particular QoS requirements [13], SDN policies support routing expressiveness well beyond the IP prefix destination level. Flows with the same IP prefix destination may be matched by distinct SDN policies, each with a different forwarding action (e.g., one policy for HTTP traffic and the other one for SSH traffic). This results in some (or all) traffic being *deflected* from its intended BGP path.

An example of an SDN-induced forwarding loop is shown in Fig. 1. Two networks A and B (N and M) connect at SDX_1 (SDX_2). A network Z originates an IP prefix to which all networks aim at sending traffic. Green links represent BGP paths. BGP policies are such that network A steadily forwards traffic through path

---

[1] All proofs are in the companion technical report of our paper [16].

[2] Any stability aspect of the global Internet BGP policies is orthogonal to the scope of this paper. Also, past studies have shown that popular BGP destinations are remarkably stable [17].
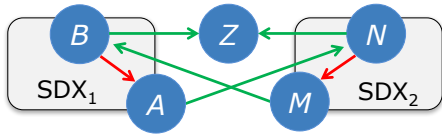
Figure 1: Example inter-SDX network.

(A N Z) while M uses (M B Z). Network B and N install two SDN policies (depicted in red) to deflect their HTTP traffic (i.e., TCP port 80) towards neighbors A and M, respectively (the order in which the rules are installed is irrelevant to the creation of a forwarding loop). This operation has a severe consequence: all HTTP traffic is now forwarded through B, A, N, M, and back to B, a forwarding loop!

Detecting deflection-induced loops is difficult as deflections are not propagated through BGP. Neither B nor N is aware of the BGP deflections installed by the other and both believe the deflected path is loop-free.

The problem of detecting deflection-induced forwarding loops is further hindered by the fact that, to quote [4], *"[networks] are not keen on sharing detailed information about their policies"*. A recent survey [6] confirmed these concerns. For this reason, solutions to detect forwarding loops without disclosing routing policies have been highly sought after in the recent past [1, 4, 7]. The most notable one, i.e., SIDR [4], advocates for a system where SDX members share with other SDXes information about the possible sets of forwarding actions of all the SDN policies affecting the same specific IP prefix without disclosing the type of traffic (e.g., whether it is HTTP or SSH) that is being deflected or, in other words, without sharing the SDN match filters. This solution preserves the privacy of the match part of the SDN policies at the price of reporting non-existing forwarding loops (false positives). For instance, if B's and N's deflections only affect HTTP and SSH traffic, respectively, no forwarding loop is created. In Sect. 4, we show that over one out of four forwarding loops reported by SIDR are incorrect. Our system aims to both accurately detect forwarding loops and improve the privacy of the SDN policies.

Safe deflections exist! One may ask if it is possible to install BGP deflections in such a way that Internet routing is loop-free without requiring any coordination among ASes. It is well-known that when BGP policies are configured according to traditional customer-provider economic relationships, as defined by the well-known Gao-Rexford conditions [11], Internet routing converges to a loop-free forwarding state. We generalize this to include BGP deflections created by SDN policies.

THEOREM 2.1. *If the BGP policies and deflections satisfy the Gao-Rexford conditions, then the forwarding state is loop-free.*

While Gao-Rexford conditions are often implemented in practice, there are unfortunately many exceptions to them such as partial transit relationships [12]. Moreover, customer-provider relationships are generally confidential. Thus, we need ways to detect potential forwarding loops before deflections are installed.
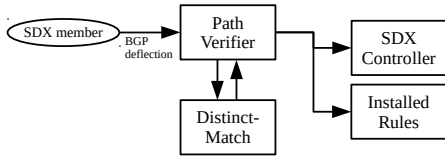
3 PRELUDE

We are now ready to discuss the system requirements that an Internet loop-detection system should meet to be of practical interest. We will then explain how we carefully design PRELUDE to tackle the challenges of satisfying such requirements and carry out loop detection in a distributed manner (Sect 3.1), and present the privacy-preserving primitive DISTINCT-MATCH for detecting overlaps of SDN rules (Sect 3.2).

System requirements. Based on the discussion from Sect. 2, we argue that any mechanism for detecting deflection-induced forwarding loops among SDXes should satisfy the following four requirements: *(i) safety*, i.e., no packet is forwarded through a cycle, *(ii) privacy*, i.e., SDN policies should be kept private to the largest possible extent and only information that is strictly necessary to ensuring the safety of routing or available through BGP should be revealed, *(iii) effectiveness*, i.e., reject as few safe SDN policies as possible (ideally zero), and *(iv) performance*, i.e., forwarding loops should be detected as soon as possible. We make extensive use of recent advances in computation over encrypted data (i.e., SMPC) to achieve these four properties.

Threat model. We rely on the *curious-but-honest* security model, in which an attacker aims to learn as much information as possible about an SDX member's policies from the exchanged messages while adhering to the agreed protocol. This assumption resonates well in today's IXP ecosystem [7] as SDX members already trust their IXPs/SDXes for forwarding their traffic to the intended destinations. An *attacker* is either an SDX operator or an SDX member who wants to learn about the SDX policies of a member of another SDX. We assume that the attacker does not gain control of any two SDXes, which are independent organizations. SDXes may learn about the IP prefix destinations of SDN policies installed at remote SDXes that deflect traffic through them but nothing more (neither who installed them nor the type of traffic involved). These privacy assumptions are stronger than the ones defined in SIDR [4] as the actions of the SDN policies are hidden from third parties.

Root-causes of forwarding loops. We observe that only two types of operations can trigger a forwarding loop: *(i)* installation/removal of an SDN policy or *(ii)* BGP updates to the routing state (e.g. a path change/withdrawal).

Figure 2: Prelude overview. When a request is received, Prelude verifies the safety of the deflected path. If activation is allowed, the rule is saved in the local table.

3.1 Overview

Fig. 2 shows the high-level architecture of PRELUDE.

The path verifier. When a member of an SDX installs/ removes an SDN policy or there is a BGP update that impacts a deflection, the member sends a request to the SDX where the rule is being installed. This request is received by the *path-verifier* running at the SDX connected to the issuing member. The path-verifier is responsible for determining which SDXes are traversed by a given path and how a path would be deflected.

To function properly, the path-verifier relies on a primitive called DISTINCT-MATCH. This primitive will indicate whether two distinct SDN rules (or policies) affect the same traffic. Through DISTINCT-MATCH, the SDX can verify, for a given flow of packets and associated path, if any other SDX along the path is deflecting the traffic. If a deflection occurs, the system will verify whether we are closing a loop. DISTINCT-MATCH is implemented in a privacy-preserving way, and the SDX can verify the traffic path without learning *anything* about the SDN policies of the requesting member.

All SDXes contacted during the verification will record the installation of a rule and notify the installing SDX of future downstream changes in policies or BGP state.

Interactions with the SDX controller. PRELUDE interacts with an SDX *controller*, which is responsible for installing SDN policies. PRELUDE filters unsafe policies before they get installed into the SDX forwarding tables.

Provable effectiveness guarantees. A common case is to have just two SDXes announcing the same prefix and potentially exchanging traffic with each other. In this case, our system guarantees provable effectiveness, as formalized in Theorem 3.1. If more than two SDXes are involved, false positives can occur (more in Sect. 4).

THEOREM 3.1. Prelude *detects a forwarding loop between two ASes at two SDXes if and only if a forwarding loop between the two ASes exists. Moreover, the members' SDN policies are not revealed neither among SDXes or any unintended member.*

Example. In the example topology from figure 1, suppose that AS B has already installed a policy r_B at SDX$_1$. AS N requests the installation of policy r_N at SDX$_2$, using BGP path (M B Z). Upon reception of this request, the path-verifier at SDX$_2$ discovers the presence of SDX$_1$ on the path. DISTINCT-MATCH is called to discover if r_N conflicts with any policy at SDX$_1$ (i.e. the set of packets matched by the policies is non-empty). If that is the case, SDX$_1$ notifies SDX$_2$ that the next deflection on the deflected path is AS N at SDX$_2$. The loop between SDX$_1$ and SDX$_2$ is detected, and the rule is rejected.

3.2 The Distinct-Match primitive

We assumed that the DISTINCT-MATCH primitive used by the path-verifier to detect if two rules will affect the same traffic can be implemented *without* revealing to that SDX the rule identifying the traffic flow. We now provide some essential background on SMPC, then describe the specific rule representations and SMPC "circuits" we used to *efficiently* implement DISTINCT-MATCH.

Secure Multi-Party Computation. SMPC is a cryptographic technique that allows multiple parties to jointly compute the outcome of a function f while keeping their respective inputs (and outputs) private.

In our work, we rely on two well-established SMPC protocol implemented in the ABY [10] framework. The first one is Goldreich-Micali-Wigderson (GMW), in which the core idea is to evaluate the successive gates sequentially. The second is the Yao's garbled circuits protocol, which relies on garbled truth tables for the circuit gates. Both protocols have different performance behavior that are discussed in Sect. 4.

In both protocols, the function f is represented as a boolean circuit consisting of basic AND and NOT gates that is jointly evaluated by *two* non-colluding entities. The inputs to the circuit are XOR-encrypted before being shared by the parties, thus ensuring perfect privacy. In our model, each of the non-colluding SMPC entities executing the protocol is an SDX.

Efficiency requires that the SMPC function be kept as simple as possible to limit the number of rounds in which entities need to exchange information. To maintain practical runtimes, we limited our use of SMPC to *overlap detection*, as described next.

Distinct-Match SMPC execution. Consider SDX$_1$, that wants to verify whether a rule r_A overlaps with a remote rule r_B installed at a remote SDX$_2$. Each SDN policy (or rule) is encoded as a string in which each bit has three possible values: 1, 0 or x (don't care). Two rules are said to be *distinct* or *non-overlapping* if there exists a bit index such that one rule matches a 1 and the other rule matches a 0. In SMPC, we model each rule r_i as a bit pattern p and a bit mask m. In the mask, the i-th bit m_i is 1 if r_i is different from x, 0 otherwise. In the pattern, the i-th bit p_i has the same value as r_i, unless r_i is x, in which case it can be any value.
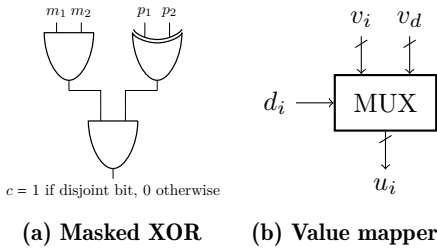
(a) Masked XOR   **(b) Value mapper**

Figure 3: SMPC boolean circuits of Distinct-Match.

		Number of rules				Baseline
Delay		1	50	500	5000	5000
1 ms	setup	2.75	12.6	94.5	1033	—
	online	4.63	10.2	30.5	151	2
10 ms	setup	20.7	48.4	162	1086	—
	online	40.6	64.2	136	382	20
100 ms	setup	201	408	1237	6061	—
	online	401	604	1395	2638	200

Table 1: Average computation time (in ms) for the comparisons. Blue (Green) cells use Yao (GMW).

The SMPC execution consists of three phases: input generation, circuit execution, and output reconstruction. Each SMPC input consists of two shares obtained by XOR-encoding the input with a random nonce: one share is the random nonce and the other is the encoded string. Without both shares, it is impossible to learn anything about the actual input. Each SDX sends to the other SDX only one share of the input.

Figure 3a shows the boolean circuit implementing the SMPC function that compares the rules r_1 and r_2. For each bit, this circuit returns 1 if the bits of the corresponding index in each rule match $0, 1$ or $1, 0$, respectively (i.e., if the rules are disjoint). We then connect all the bits with an OR gate to return a single (encrypted) bit for the comparison.

If the rules are non-distinct (i.e., some packets would be matched by both rules), it means that the remote SDX already has an active overlapping rule that would deflect some traffic of the new SDN rule. In that case, the circuit returns the identifier of the next SDX in the deflection path created by the rule installed at SDX_2. If the rules are distinct, a special value is returned as illustrated in Figure 3b (v_i is the next-hop identifier for rule i, and v_d stands for "dummy"). The SDXes finally exchange their shares and reconstruct the output.

To make the verification efficient, Distinct-Match compares multiple (potentially thousands of) rules in parallel and returns a set of all the next-hops created by the active rules. The output of the circuit is randomly and secretly shuffled to protect the remote SDX's rules.

Section 4 explains the cost of using this primitive when more than two SDXes are involved in a loop.

Implementation. We implemented and evaluated the above boolean circuits using the ABY framework [10], a C++ toolbox supporting various implementations of SMPC. The circuits can be evaluated with either the GMW or Yao's garbled circuit protocol. Both protocols return the same result, but have performance differences that are discussed in Sect. 4.

Because the bottleneck for SMPC circuit evaluation is the one-way communication delay (and not the local computation), the queries to Distinct-Match are done in parallel for each SDX — only successive deflections need to be discovered sequentially. This means that the cost to verify the safety of a new rule is equal to the cost of the longest path in the deflection graph.

4 EVALUATION

We now assess the effectiveness of Prelude in terms of its ability to correctly identify safe or unsafe SDN policies. Prior to that, we show the results of our micro-benchmarks for the Distinct-Match primitive in terms of processing time for different number of rules, which is the costliest part of our system. We finally expand our evaluation to the performance of the whole system. We also investigate how different types of BGP deflections may affect the length of the routing paths.

Distinct-Match micro-benchmarks. We evaluate the performance of Distinct-Match. In this micro-benchmark, two SMPC entities jointly evaluate the Distinct-Match circuit. One SMPC entity holds *one single* SDN rule and the other holds a *set* of SDN rules. We study the performance of our Distinct-Match implementations along three different dimensions: *(i)* the time to evaluate the SMPC circuit, *(ii)* the communication delay between the two entities, and *(iii)* the size of the set of SDN rules. We break down evaluation time by *setup* time, which is a phase of SMPC computation that can be pre-computed and depends only on the maximal number of possible rules, and *online* time, which depends on the actual inputs and directly affects the system reactiveness.

Table 1 shows the average execution time over 50 executions of the Distinct-Match operation for different communication delays and sizes of the set of rules. In all experiments, we set the length of the rules to 13 bytes, which is sufficient to encode source/destination IPv4/TCP addresses/ports. The baseline is $2 * RTT$, ignoring any local computation cost. Evaluating the circuit with Yao's garbled circuit protocol (blue cells) is more efficient for small numbers of rules but scales linearly in the number of rules. Using the GMW protocol (green cells) scales logarithmically with the number of rules and, depending on the delays, is more efficient when we compare hundreds or thousands of rules at once. We also observe that the online time is directly
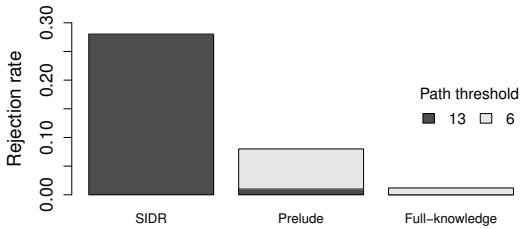
Figure 4: Comparison of rejected correct rules.

proportional to the round-trip delay, clearly indicating that the communication delay among the two SMPC entities is the limiting factor that prevents achieving even more performing SMPC executions.

Loop-detection effectiveness. We now compare the ability of PRELUDE to effectively installs as many SDN policies as possible with SIDR [4], the state-of-the-art mechanism. To evaluate it, we used the same simulation technique from SIDR. The topology is built from the CAIDA AS graph [5] augmented with the links from the combined IXP dataset. It computes paths to 1000 randomly selected AS destinations. Each of the 421 IXPs is considered to be an SDX. Moreover, we assume that traffic traverses an SDX if its path traverses two members of that IXP one after the other.

Each SDX member generates between one and four policies towards 20% of the other members (at most 50 members). Those policies are restricted to matching the IP protocol (TCP or UDP) and a random source or destination port. Unlike SIDR, we select the source and destination port randomly. This represents the fact that no rule will be present for the most commonly used ports, as the policies for those are implemented through BGP. This also avoids creating duplicate rules. As a consequence, the number of created loop is significantly lower and those loops are often more complex than those considered in the SIDR evaluation. A policy from member A to member B is applied to each route announced by B.

Figure 4 shows the results for three loop detection systems: SIDR, PRELUDE, and a perfect knowledge approach based on path exploration without any privacy restrictions, which we use as an ideal upper bound for our approach. We look at a path exploration dimension called *path-threshold*, which caps the maximum number of deflections to be followed by a path exploration mechanism. We observe that the false positive rate of SIDR in our setting (random ports) is at 28% while the false positive rate of the perfect knowledge solution is 0% (with a path-threshold of 13).

We observe that PRELUDE sharply reduces the number of incorrectly detected loops when compared to the less privacy-preserving SIDR. With a path-threshold of 6
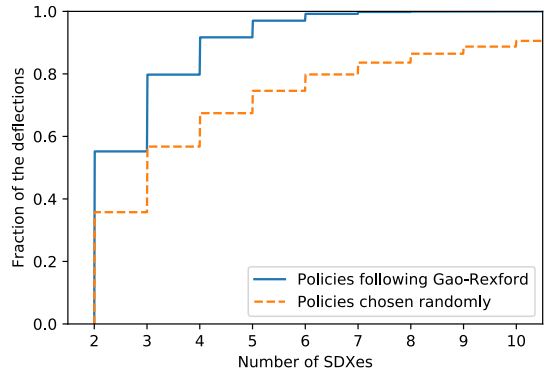

Figure 5: Number of SDXes in the deflected paths.

SDXes, PRELUDE has only 8% of false positives, which drops to 0.3% when the path-threshold is set to 13.

Path exploration performance. The time required for verification is proportional to the number of SDXes traversed by the deflected path and the DISTINCT-MATCH execution time. While exploring the network hop-by-hop is normally costly, real paths generally cross only a few ASes [3], and similarly only cross a very limited number of SDXes. We evaluated the number of SDXes crossed by deflected paths and report the CDF in Figure 5. We consider two classes of BGP deflections: Gao-Rexford (GR) compliant ones [11] (blue solid line), i.e., an AS never deflects traffic to a provider if a customer route is available and non-compliant ones (orange dashed line), chosen randomly. We observe that GR conditions keeps the number of SDXes in the path low (60% of the paths with only two SDXes). Yet, GR conditions are deemed non realistic in practice as exceptions may exist. The second scenario represents a worst-case situation, where ASes deflect traffic through any random route they learned, resulting in higher path inflation. We observe that the SIDR assumption about the presence of an SDX in a path is an over-estimate on the number of traversed SDXes. When only two SDXes are traversed by a deflected path, the verification cost is exactly equal to the one of executing DISTINCT-MATCH.

5 CONCLUSION

In this paper, we focused on reconciling SDXes with the limited BGP routing expressiveness and ASes' natural desire for preserving the privacy of their business information. We designed and implemented a general privacy-preserving primitive for detecting overlaps among SDN rules that can be executed in practical runtimes. We then introduced PRELUDE, a framework that allows SDXes to verify the safety of SDN policies while keeping those policies private. Finally, we investigated the impact of SDN deflections on Internet routing, demonstrating positive conditions for loop-free Internet routing.

Acknowledgements

REFERENCES

[1] G. Asharov, D. Demmler, M. Schapira, T. Schneider, G. Segev, S. Shenker, and M. Zohner. Privacy-Preserving Interdomain Routing at Internet Scale. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 3, 2017.

[2] AtlanticWave-SDX: A Distributed Intercontintal Experimental Software Defined Exchange for Research & Education Networking, 2015. https://itnews.fiu.edu/wp-content/uploads/sites/8/2015/04/AtlanticWaveSDX-Press-Release_FinalDraft.pdf.

[3] AS65000 BGP Routing Analysis. https://bgp.potaroo.net/as2.0/bgp-active.html.

[4] R. Birkner, A. Gupta, N. Feamster, and L. Vanbever. SDX-Based Flexibility or Internet Correctness? Pick Two! In *SOSR*, 2017.

[5] The CAIDA AS Relationships Dataset, 2015. http://www.caida.org/data/as-relationships/.

[6] M. Chiesa, D. Demmler, M. Canini, M. Schapira, and T. Schneider. Internet Routing Privacy Survey, 2017. http://bit.ly/2rjT7Nj.

[7] M. Chiesa, D. Demmler, M. Canini, M. Schapira, and T. Schneider. SIXPACK: Securing Internet eXchange Points Against Curious onlooKers. In *CoNEXT*, 2017.

[8] M. Chiesa, C. Dietzel, G. Antichi, M. Bruyere, I. Castro, M. Gusat, T. King, A. W. Moore, T. D. Nguyen, P. Owezarski, S. Uhlig, and M. Canini. Inter-Domain Networking Innovation on Steroids: Empowering IXPs with SDN Capabilities. *IEEE Communications Magazine*, 54, Oct. 2016.

[9] DE-CIX: Project Endeavour, 2015. https://www.de-cix.net/en/about-de-cix/research-and-development/endeavour.

[10] D. Demmler, T. Schneider, and M. Zohner. ABY: A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*, 2015.

[11] L. Gao and J. Rexford. Stable Internet Routing Without Global Coordination. In *SIGMETRICS*, 2000.

[12] P. Gill, M. Schapira, and S. Goldberg. A Survey of Interdomain Routing Policies. *ACM SIGCOMM Computer Communication Review*, 44(1), Dec. 2013.

[13] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever. An Industrial-Scale Software Defined Internet Exchange Point. In *NSDI*, 2016.

[14] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying Network-Wide Invariants in Real Time. In *NSDI*, 2013.

[15] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[16] Prelude Technical Report. https://arxiv.org/abs/1806.09566.

[17] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP Routing Stability of Popular Destinations. In *SIGCOMM Workshop on Internet Measurements*, 2002.

[18] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford. Central Control Over Distributed Routing. In *SIGCOMM*, 2015.