SDX-Based Flexibility or Internet Correctness? Pick Two!

Rüdiger Birkner[◇], Arpit Gupta[⋆], Nick Feamster[⋆], Laurent Vanbever[◇]
[◇]ETH Zürich [⋆]Princeton University

Abstract

Software-Defined Internet eXchange Points (SDXes) are recently gaining momentum, with several SDXes now running in production. The deployment of multiple SDXes on the Internet raises the question of whether the interactions between these SDXes will cause correctness problems, since SDX policies can deflect traffic away from the default BGP route for a prefix, effectively breaking the congruence between the control plane and data plane. Although one deflection on a path will never cause loops to occur, combining multiple deflections at different SDXes can lead to persistent forwarding loops that the control plane never sees.

In this paper, we introduce *SIDR*, a coordination framework that enables SDXes to verify the end-to-end correctness (*i.e.*, loop freedom) of an SDX policy. The challenge behind *SIDR* is to strike a balance between privacy, scalability, and flexibility. *SIDR* addresses these challenges by: *(i)* not requiring SDXes to disclose the flow space their SDX policies act on, only the next-hop they deflect to; and *(ii)* minimizing the number of SDXes that must exchange state to detect correctness problems. *SIDR* manages to preserve the flexibility of SDX policies by activating the vast majority of the safe policies, the policies that do not create a loop. We implemented *SIDR* on the SDX platform and showed its practical effectiveness: *SIDR* can activate 91% of all safe policies while preserving privacy and scalability and can perform correctness checks in about one second.

CCS Concepts:
Networks → Network architectures; Programmable networks;

Keywords:
Software Defined Networking (SDN); Internet Exchange Point (IXP); Routing

SOSR'17, April 03–04, 2017, Santa Clara, CA, USA

DOI: http://dx.doi.org/10.1145/3050220.3050221

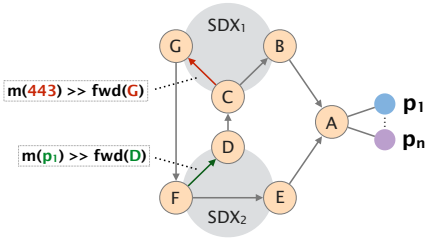


Figure 1: *IP traffic for AS A is caught in a persistent forwarding loop due to the SDX peer C at SDX_1 and F at SDX_2 independently deflecting traffic from the announced BGP path via SDX policies.*

1 Introduction

Software-Defined Internet eXchange Points [13, 14] (SDXes) are seeing increased deployment on the Internet. Not only have multiple SDXes been deployed [1, 3], but also the two largest Internet eXchange Points (IXPs) have begun preliminary trials of the SDX [2, 15]. SDXes bring flexibility to an aging Internet routing system by enabling members to flexibly override their default BGP route using fine-grained SDX policies. These policies enable new traffic-management capabilities such as improved inbound traffic engineering or application-specific peering.

Problem: Unfortunately, by silently overriding the default BGP paths, SDXes lead to a phenomenon known as *deflections*. Intuitively, a deflection arises whenever the actual forwarding path for a destination does not match the path computed by the control plane (here, BGP): the two planes are not congruent anymore. Because one of the roles of BGP is to guarantee the eventual correctness of the forwarding plane, bypassing it can result in forwarding anomalies, including forwarding loops.

Figure 1 illustrates a simple situation where SDX policies introduced at two SDXes (SDX_1 and SDX_2) lead to forwarding loops. SDX_1 interconnects B, C, and G, while SDX_2 interconnects E, D, and F. A advertises n IP prefixes $(p_1, ..., p_n)$. The preferences are such that D prefers the paths from C while G prefers the paths from F. As dictated by BGP, whenever an AS receives multiple equally preferred paths for the same destination, it breaks the tie by preferring shorter AS paths. Two SDX policies are defined: *(i)* F deflects all traffic destined to prefix p_1 to D, while C deflects all HTTPS-traffic by defining a policy matching on the destination port. The end result is a *persistent forwarding loop* for all packets entering C, D, F or G with a destination IP in p_1 and TCP port 443.

Passively or actively detecting such SDX-induced loops is challenging (and often impossible) for two reasons: First, the loops are not visible to the BGP control plane, making it impossible to detect these loops by monitoring BGP routes. In this example, D believes that HTTPS traffic directed to p_1 will go via $[C,B,A]$ (according to the AS path) while it ends up traversing $[C,G,F]$. Second, any subset of the flow space—for any of the 600,000s+ Internet prefixes [4]— can be caught in an SDX-induced loop, making active detection (e.g., via probing) ineffective as well. In our example, only the traffic for port 443 and prefix p_1 loops, while the rest of the traffic successfully reaches the destination.

This work: In this paper, we present *SIDR* (Safe Inter-domain Deflection-based Routing), a new coordination framework that enables SDXes to verify the end-to-end correctness of an SDX policy. Given the difficulty of detecting SDX-induced loops, *SIDR* adopts a proactive approach by having SDXes exchange information about the deflections they introduce. Doing so, *SIDR* restores the congruence between the SDN data plane and the BGP control plane and lets each SDX check for policy correctness locally.

Challenges: Although exchanging information can guarantee the absence of loops, it also introduces three challenges that are seemingly at odds: *(i)* privacy, revealing as little information about the policies (flow space and actions) as possible; *(ii)* scalability; while preserving *(iii)* flexibility, allowing to activate as many safe policies as possible. The first two call for exchanging as little information as possible. Indeed, SDXes and their participants are not keen on sharing detailed information about their policies. At the same time, exchanging detailed and highly volatile information between many entities would not scale. Yet, sharing little information contradicts with the flexibility objective. Intuitively, with fewer information, *SIDR* can check the correctness of fewer policies forcing it to block safe SDX policies.

SIDR addresses the needs for privacy and scalability, while preserving most of the SDXes' flexibility. In terms of privacy, *SIDR* only requires SDXes to exchange information about *where* they deflect traffic to, not *what* is being deflected there. As such, SDXes do not have to share information about the most sensitive part of their policies: the match field. In terms of scalability, *SIDR* minimizes the number of SDXes that have to exchange information. By relying on eventual consistency (like BGP), *SIDR* also enables SDXes to rapidly activate a policy. In terms of flexibility, *SIDR* still enables 91% of the safe policies to be activated. In contrast, only 2% of the safe policies can be activated without *SIDR*.

Novelty: Previous work such as Veriflow [17] and HSA [16] also aim to guarantee loop-free forwarding. Yet their detection and prevention techniques are geared at single domains where the entire forwarding state is known. In contrast, *SIDR* considers *interdomain* correctness with *partial information*. Although the SDX setting motivated our work on *SIDR*, its principles are general and can be applied to ensure the correctness of other deflection frameworks such as Path Splicing [20] or TED [24].

Main contributions: Our main contributions are:

- An introduction to the problem of SDX-induced forwarding loops along with the difficulty of avoiding them (§2).
- A new coordination framework, *SIDR*, which enables to guarantee Internet correctness in the presence of SDX-induced deflections (§3). *SIDR* reduces the amount of information to be shared and the number of SDXes sharing it, while preserving high-levels of flexibility.
- A complete implementation of *SIDR* atop the existing SDX platform along with a comprehensive evaluation showing that *SIDR* scales and preserves flexibility, being able to activate 91% of all safe policies (§4).

2 Interdomain Deflections at SDXes

In this section, we highlight the conditions for SDX-induced forwarding loops to occur (§2.1). As the problem lies in the incongruity of the data and control plane, we introduce a simple solution that requests the SDXes to share all their state (§2.2). While this solution is correct, it is not realistic. We explain in the next section how SIDR improves on the strawman solution to make it practical.

2.1 SDX-induced Forwarding Loops

As not all SDX policies create a forwarding loop, we now look at the reason for loops to exist to better understand the problem. Specifically, two *necessary and sufficient* conditions must be met:

1. There must be at least two SDX policies defined at two or more SDXes, and the concatenation of the AS paths of the routes used contains a loop;
2. The affected flow spaces of these SDX policies have a nonempty intersection.

The two conditions are *necessary*. Indeed, a single SDX policy cannot create a loop since the AS path which the deflection tries to use is loop-free thanks to BGP's loop detection algorithm. Moreover, even when considering multiple SDX policies, if their flow spaces do not intersect, no traffic will ever match all these policies at the same time.

The two conditions are also *sufficient*. Whenever we have two or more SDX policies (deflections) that form a loop, and the flow spaces of these SDX policies are intersecting, it creates a persistent forwarding loop.

2.2 Strawman Solution

A simple solution resynchronizes the data and control planes. Unfortunately, whether a given SDX policy creates a loop or not depends not only on itself, but on all the other policies already present at other SDXes. We now describe a strawman approach in which all SDXes instantaneously share *all* their information, including their SDX policies and the content of their BGP routing table (RIB). This sharing is done using a strongly consistent, distributed database.

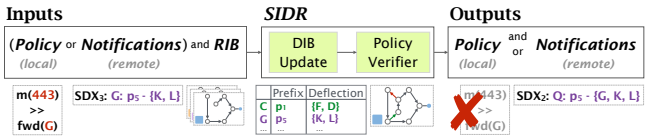


Figure 2: *For every SDX policy activation request and every change in the topology,* SIDR *checks whether a forwarding loop exists and takes counter-measures if necessary.*

Before installing a new policy (i.e., a new deflection), each SDX analyzes the AS path of the route the policy tries to use by looking for SDXes on the deflected path. For all SDXes on the deflected path, it checks whether at least one of them has a policy installed whose flow space intersects with that of the new policy. If so, it analyzes the deflected AS path of that policy and continues until: *(i)* the flow spaces do not intersect anymore; *(ii)* the destination is reached; or *(iii)* a loop is detected. One can think of this process as a DFS traversal. In the first two cases, the SDX activates the policy and shares this information with all the other SDXes. In the third case, it will not activate the policy as it creates a forwarding loop.

2.3 Challenges

As we highlighted earlier, the challenge in designing an effective and practical system is to strike a balance between *(i)* privacy, *(ii)* scalability, and *(iii)* flexibility. The strawman solution described above prioritizes high flexibility, *i.e.* ensuring as many SDX policies are activated as possible. However, this high flexibility comes at the cost of poor scalability and privacy—making it an impractical solution to deploy. We will now describe the scalability and the privacy challenges in greater detail :

Scalability: Two dimensions have to be addressed: *(i)* the size of the state the system needs to maintain and exchange, and *(ii)* the rate at which updates need to be processed. Synchronizing the RIBs and activated policies amongst all SDXes is non-trivial considering the size of the Internet. Similarly, processing this information to detect forwarding loops is challenging as many prefixes can be affected at the same time and the state is volatile. Fast processing is imperative to counter forwarding loops in a timely matter.

Privacy: As SDX forwarding policies can be defined on arbitrary flow space, not just on the destination prefixes, SDXes are required to share the full policy information with each other to perfectly determine the safety of a policy. In practice, participants at these IXPs are reluctant to share their fine-grained SDX policies.

3 *SIDR*

SIDR addresses the challenges we highlighted in the previous section by exchanging only basic information about the policies, minimizing the number of SDXes required to exchange information with and basing the loop detection on partial information. Below, we present in §3.1 an overview of *SIDR*, a system that coordinates the activation of SDX policies using local instances at SDXes, and demonstrate its principles

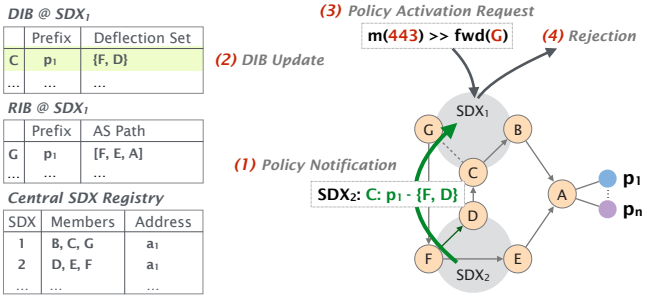


Figure 3: *Overview of the events at SDX_1 due to a successful policy activation at SDX_2 and a local policy activation request.*

with an example. Then, we describe the three components of *SIDR*: the communication manager (§3.2), the deflection information base (§3.3) and the policy verifier (§3.4).

3.1 Overview

SIDR is a daemon running on top of every SDX consisting of three major components: *(i)* the *communication manager*, which manages the inter-SDX communication; *(ii)* the *deflection information base (DIB)*, which provides a local view of the policies activated locally and at remote SDXes; and *(iii)* the *policy verifier*, which processes all policy activation requests and constantly verifies the locally activated policies.

Figure 2 provides an overview of the system. *SIDR* takes local policy activations, deflection notifications and changes in the RIB as input, updates the DIB, and starts the policy verification process to determine whether the resulting forwarding state is loop-free. In case of a policy activation request, *SIDR* grants (or denies) permission to activate the new SDX policy and informs the other SDXes. If an already activated policy suddenly causes a loop because of a RIB change or remote policy activation, it must be removed.

After this broad overview, we give an intuition of how *SIDR* operates using the example from §1 and Figure 3. We take a closer look at the *SIDR* instance running at SDX_1. SDX_2 installed a policy from F to D for all traffic destined to p_1. *(1)* It sends a deflection notification to the next affected SDX, SDX_1. This notification contains the following information: *(i)* the affected prefixes: here p_1; *(ii)* the identifier of the SDX sending the notification: SDX_2; *(iii)* the deflection set: $\{F, D\}$; and *(iv)* the ingress participant at the remote SDX: C. It informs the SDX about all the ASes from which it might receive deflected traffic. *(2)* SDX_1 inserts the entry p_1: $\{F, D\}$ into its DIB. *(3)* Later, SDX_1 receives a request from participant C to install a policy which deflects all HTTPS-traffic to participant G. The policy verifier starts a loop check and accepts the policy if no loop is created. If necessary, it crafts notifications to inform its neighboring SDXes about the new deflection. To verify the policy, *SIDR* first retrieves all the routes G advertised. In this example, it finds a route for p_1 with the following AS path: $[G, F, E, A]$. It then queries the DIB, retrieves the entry for p_1—$\{F, D\}$—and combines it with the route. It is apparent that the new SDX policy is not

safe as F appears in both the deflection set and the AS path. *(4)* Therefore, the SDX policy is rejected.

Upon reception of a deflection notification or a change in the RIB, *SIDR* runs the same verification process to check whether any of the already activated SDX policies are now creating a loop. If this is the case, *SIDR* deactivates this policy. Otherwise, it simply updates the DIB and sends the resulting notifications to its neighbors.

3.2 Communication Manager

Because a loop consists not only of local, but also remote policies, *SIDR* instances must exchange information about the activated policies amongst each other. The communication manager performs this task. Maintaining a session and exchanging information constantly with hundreds of remote SDXes around the globe does not scale. Fortunately, the majority of the remote policies do not affect a local SDX. Thus, it is not necessary to share each and every policy with all the SDXes. It suffices to establish sessions only between immediate neighbors. Here, we define neighbors of an SDX as the ones that appear first on any of the advertised paths at that SDX. To still have the information spread to all affected SDXes, each *SIDR* instance aggregates the received information and passes it on to its immediate neighbors. Information for a given prefix has only to be passed on to the neighbors on the best path and all paths for that prefix which are used by an active policy.

Two mechanisms can be used to detect the immediate neighbors of an SDX: *(i)* a specialized neighbor discovery protocol, or *(ii)* a centralized SDX registry where each SDX and all of its participants are listed. The communication managers can then combine the information from the registry with the information about locally advertised paths to infer their immediate neighbors.

3.3 Deflection Information Base

To check whether a policy creates a loop, *SIDR* must know about deflections that result from remote policies. The SDXes exchange this information through the communication manager, store it in the DIB and pre-process it for the policy verifier.

Each *SIDR* instance maintains its own local view of all the locally relevant policies in the Internet. These local views facilitate faster processing of policy activation requests, because the system does not have to rely on remote SDXes. This decision creates the risk of transient loops, since the local DIB might be out-of-sync with the actual network state. Fortunately, this behavior is temporary and as soon as the local view is in-sync again, the forwarding loops are detected and eliminated—ensuring eventual loop-freedom.

The DIB has three tables: *(i)* the input table: which stores all the incoming deflection notifications; *(ii)* the local table: which stores the information needed for the policy verifier. For each SDX participant and prefix, the DIB contains the set of ASes from which deflected traffic might be received. This information is built by combining the entries of the input table. For example, consider the case where we have two entries in the input table for a prefix p and participant A: One from SDX_1 and one from SDX_2. We combine these two entries by building the union of their deflection sets before storing it in the local table; and *(iii)* the output table: which stores the notifications sent to neighbor SDXes. These notifications do not only contain information about deflections caused by local policies, but also about all relevant remote deflections. Hence, they are crafted based on the locally activated policies and the entries in the local table. Assume,the following entry is in the local table: C: p_1 - $\{F,D\}$ and C has an active policy for prefix p_1 towards B: $\{B,C\}$. Then, we combine the two sets, and the resulting deflection set is $\{B,C,F,D\}$.

3.4 Policy Verifier

The most important part of the system is the policy verifier. It is in charge of performing the loop check whenever a local participant wants to activate a new SDX policy, a deflection message is received or the RIB changes.

To address both the scalability and privacy challenges, we base the entire loop detection only on partial information: For each policy, we consider only the affected prefixes and the two ASes between which it is activated, and completely disregard the exact flow space it is defined on. This choice not only keeps the exact policies private, but also reduces the information that needs to be exchanged. However, it comes at the cost of false positives as not all loops can be detected precisely and *SIDR* has to be conservative to not risk correctness.

To determine whether a given policy, active for some prefixes, creates a loop, the following information is needed: *(i)* all the RIB entries (AS path) and *(ii)* all the entries from the local table of the DIB (deflection set) for these prefixes. To simplify the verification, we group all prefixes with the same AS path and deflection set. Hence, for every prefix group, the policy verifier compares the ASes on the AS path, to the ASes in the deflection set. The ASes on the AS path are the ones being potentially traversed due to the policy, while the deflection set contains the ASes through which the traffic might arrive. If an AS appears both in the AS path and the deflection set, it is possible that a forwarding loop exists.

To speed up the policy verification process, *SIDR* precomputes a so called forbidden set which stores for each pair of participants and prefix group whether a policy might create a loop. This can be done offline as the exact flow space of an SDX policy is not required for the loop check. Whenever the DIB or the RIB changes, we run the loop verification mechanism for all affected participant pairs and prefixes. At the same time, the forbidden set is updated. Upon a policy activation request, the policy verifier does not need to consult the RIB and DIB, but can simply check the forbidden set.

Because policy activation relies on the DIB, policies might be activated concurrently at multiple SDXes such that their combination creates a forwarding loop. The SDXes will discover this loop after all the deflection notifications have spread, which can in turn cause an oscillation as the SDXes

repeatedly deactivate and activate the conflicting policies. To avoid such oscillations, we add a timestamp and a random number to the notifications. In such a case, the SDX with the youngest deflection and, in case of equal timestamps, the highest number has to backoff and deactivate it. If the random numbers are equal, all SDXes remove the policy. This tiebreaker guarantees that the framework always converges to a stable state as only a single SDX needs to deactivate its policy to break the loop.

4 Evaluation

We now demonstrate the flexibility and the processing rate of *SIDR*. The evaluation has two parts: *(i)* large-scale simulations evaluating how many safe policies *SIDR* activates (§4.1) and *(ii)* a set of microbenchmarks evaluating *SIDR*'s raw performance in terms of: policy activation requests, deflection notifications and RIB updates (§4.2).

Our results show that *SIDR* can safely activate more than 90% of the policies and that correctness checks can be handled in no more than a few seconds.

4.1 *SIDR* Privacy and Flexibility

We evaluate how much the amount of information revealed (privacy) influences the number of safe policies activated (flexibility). Specifically, we simulate *SIDR* and three other schemes with different levels of information exchange on the AS graph from CAIDA from 2015-10-01 [5].

It is well-known that inferred AS-level topologies miss the majority of IXP links [7]. Therefore, we augment the graph with IXP links using the combined IXP dataset [18]. Our augmented AS-level topology has a total of 52,040 nodes (of which 421 are IXPs) and 1,251,811 edges (of which 1,053,654 edges result from the augmentation). Using this graph, we then compute the paths from every AS to a randomly selected group of 1,000 ASes according to the routing tree algorithm from [11]. We consider each of the 421 IXPs to be an SDX. Moreover, we consider that a given SDX is crossed by some traffic for a destination whenever two consecutive ASes in the AS path for the destination are members of that SDX.

At each SDX, we generate between one and four policies per participant towards 20% (at most 50) of the other participants. We select the flow space for each policy according to a port distribution observed in real traffic traces [6]. Hence, the well-known ports (e.g., TCP 80 or UDP 53) are preferably chosen. The activation requests arrive one after the other.

We evaluate the proportion of *safe policies* that can be activated considering four information exchange schemes: *(i)* ∅: in which the only available information is the local RIB and whether a specific AS path crosses an SDX. Hence, whenever a policy aims to deflect on a path that crosses another SDX, we consider it to be unsafe and reject it; *(ii) no prefixes*: a *SIDR*-like system which does not exchange the actual prefixes for which the policy is active, but the participants involved; *(iii) SIDR*; and *(iv) strawman*: in which all SDXes have full knowledge of all SDX policies. We use the results of *strawman* as benchmark as it activates all safe policies (§2.2).

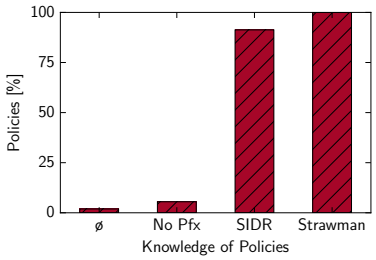


Figure 4: *Flexibility of* SIDR *compared to different levels of exchanged information.*

Figure 4 depicts the fraction of safe policies that end up being activated under the different schemes. *SIDR* manages to activate the vast majority (91%) of the safe policies, and this, without revealing any information on the flow space. In contrast, ∅ and *no prefixes* only end up activating 2% and 5% of the safe policies, respectively. The poor performance of ∅ results from the fact that IXPs often appear on paths [21] forcing the vast majority of the policies to be rejected. Finally, the poor performance of *no prefixes*—in which SDXes do exchange deflection information—is a result of the SDXes not being able to consider the per-destination routing trees but instead the AS graph itself (being prefix agnostic) to detect loops. Given how richly connected our topology is, these loops happen frequently and end up preventing the SDXes to activate most of the policies.

4.2 *SIDR* Scalability

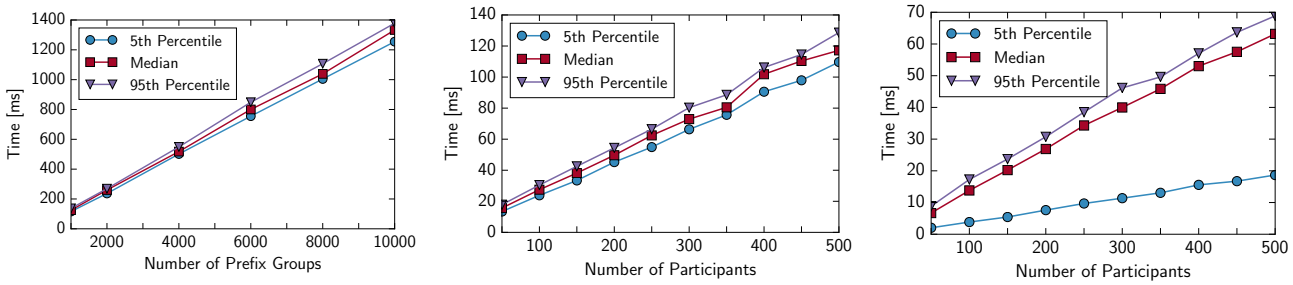
We now evaluate the performance of our *SIDR* implementation using microbenchmarks for the three main events which cause *SIDR* to trigger a loop check.

We implemented *SIDR* as a module on top of SDX-Ryu [22]. It acts as a transparent layer between the participants requesting a policy activation and the SDX-Ryu. If the policy is safe, it is passed on to the SDX. Otherwise, the policy is rejected. The module also: *(i)* handles the communication with other *SIDR* modules located in other SDXes; *(ii)* maintains the DIB; and *(iii)* hooks into the RIB of the SDX.

In the following experiments, we instantiate *SIDR* without the underlying fabric as we are only interested in measuring the raw performance of SIDR, not the actual forwarding behavior. We run all our experiments 100 times on a server with 4 physical cores at 2.4 GHz and 36GB of RAM.

Policy activation: We measure how long *SIDR* takes from the reception of a policy activation request to the decision to accept or reject it. For this, we set up two participants (ingress and egress). The egress participant advertises between $1,000$ and $10,000$ prefix groups. For each prefix group, we have with probability 0.5 a corresponding entry in the DIB.

We then consider that the ingress participant provides a single SDX policy directed to the egress participant. *SIDR* therefore needs to check for every prefix group whether a loop is created. Figure 5a shows that this processing time is linear in the number of prefix groups. Even when considering



(a) *Time to process a policy activation request.* **(b)** *Time to process a deflection notification.* **(c)** *Time to process a RIB update.*

Figure 5: *Benchmarks of the three main events: Policy activation request, deflection notification and RIB update.*

10,000 prefix groups, *SIDR* only takes 1.4 seconds to check the policy. Note that the number of prefix groups is significantly smaller than the number of prefixes advertised from the egress participant.

Deflection notification: We then measure the time it takes for our *SIDR* prototype to process a deflection notification coming from another SDX. Recall that a deflection notification triggers *SIDR* to update the DIB. Therefore, the more policies the participant—through which a notification was received—has, the more complex the processing of such a notification becomes. For this experiment, we consider that a single participant has active policies towards 50 to 500 participants which all announce the same prefix group.

Figure 5b shows that the processing time for a deflection notification is linear in the number of policies a participant has for the same prefix group. Even with active policies to 500 participants advertising the same prefix group, *SIDR* only takes 140 ms to process the notification. We note that a recent study showed that at most 25 participants announce the same prefix group [15] for which *SIDR* needs less than 20 ms.

RIB change: Finally, we measure the time it takes for *SIDR* to check the loop-freedom of all the policies affected by a RIB update. Intuitively, the more policies are affected by a route change, the longer it takes. In this experiment, we have a single participant advertising one prefix group to 50 to 500 participants which all have one policy installed. Through each of these participants one deflection notification was received. We then update or withdraw the prefix group and measure the time *SIDR* needs to process the RIB update.

Figure 5c shows that the processing time is linear in the number of participants that have a policy installed towards the participant sending the route update. Again, even when considering large SDXes with 500 participants who all happen to have a policy affected by the route change, it only takes our *SIDR* prototype 60 ms to perform the loop freedom check.

5 Related Work

SDX platforms: Multiple SDX initiatives exist [1, 3, 14]. While the scalability [13] and security [12] have been studied, none of the approaches have looked at ensuring that the resulting forwarding state is correct.

BGP Deflections: The fact that the BGP AS path and the actual forwarding path do not necessarily conform (i.e., a deflection exists) has been shown in the past [19]. Yet, with SDXes deployed, we expect the number of deflections to significantly increase motivating the need for *SIDR*. Many works have proposed to introduce deflections: Some use special headers to synchronize the data and control plane [9, 20]. Others [8, 23] craft special AS paths to enable the BGP loop detection to detect the loops. [24] relies on the fact that if all ASes follow Gao-Rexford [10], no loops can exist.

Loop Detection: [16, 17] prevent forwarding anomalies within networks and require full information. *SIDR* addresses interdomain forwarding with partial-information.

6 Conclusion

In this paper, we showed that different SDXes installing policies without coordination can lead to persistent, and particularly hard to debug, forwarding loops in the Internet. We introduced *SIDR*, a coordination framework that enables SDXes to verify the correctness of a policy. *SIDR* strikes a balance between the need for *privacy* and *scalability* while preserving much of the *flexibility* offered by SDX platforms. The key insights behind *SIDR* are to: *(i)* exchange information about *where* traffic is diverted not *what* is diverted; and *(ii)* minimizing the amount of SDXes requiring to exchange state.

We implemented *SIDR* and showed its practical effectiveness. In future work, we want to consider how *SIDR* can detect and handle misbehaving SDXes. We also want to better study the fairness aspects, making sure that each SDX has the same chance of activating a policy.

Acknowledgments

We thank Jennifer Rexford, Tobias Bühler, Florian Scheidegger and the anonymous reviewers for their constructive feedback and comments. Rüdiger Birkner was supported by the Zeno Karl Schindler foundation during his stay at Princeton University. This work was supported in part by the NSF under grant CNS-1539920.

References

[1] NZ scores first OpenFlow controlled connection to an IX, 2012. http://list.waikato.ac.nz/pipermail/nznog/2012-December/019635.html. (Cited on pages 1 and 6.)

[2] ENDEAVOUR Project Contributors, 2015. https://www.h2020-endeavour.eu/consortium-0. (Cited on page 1.)

[3] Pica8 Powers SDN-Driven Internet Exchange, 2015. http://www.pica8.com/news/pica8-powers-sdn-driven-internet-exchange/. (Cited on pages 1 and 6.)

[4] CIDR REPORT, 2016. http://www.cidr-report.org/as2.0/. (Cited on page 2.)

[5] The CAIDA AS Relationship Dataset - 20151001, 2016. http://www.caida.org/data/as-relationships/. (Cited on page 5.)

[6] The CAIDA UCSD Anonymized Internet Traces 2015 - 20150917, 2016. http://www.caida.org/data/passive/passive_2015_dataset.xml. (Cited on page 5.)

[7] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, and W. Willinger. Anatomy of a large European IXP. In *ACM SIGCOMM*, Helsinki, Finland, 2012. ACM. (Cited on page 5.)

[8] J. M. Camacho, A. García-Martínez, M. Bagnulo, and F. Valera. ASSEMBLER: A BGP-compatible Multipath Inter-Domain Routing Protocol. 2011. (Cited on page 6.)

[9] I. Ganichev, B. Dai, P. B. Godfrey, and S. Shenker. YAMR: Yet Another Multipath Routing Protocol. *ACM SIGCOMM CCR*, 40:13–19, 2010. (Cited on page 6.)

[10] L. Gao and J. Rexford. Stable Internet Routing Without Global Coordination. *IEEE/ACM ToN*, 9:681–692, 2001. (Cited on page 6.)

[11] S. Goldberg, M. Schapira, P. Hummon, and J. Rexford. How Secure Are Secure Interdomain Routing Protocols. In *ACM SIGCOMM*, New Delhi, India, 2010. (Cited on page 5.)

[12] A. Gupta, N. Feamster, and L. Vanbever. Authorizing Network Control at Software Defined Internet Exchange Points. In *ACM SOSR*, Santa Clara, CA, USA, 2016. (Cited on page 6.)

[13] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever. iSDX: An Industrial-Scale Software Defined Internet Exchange Point. In *USENIX NSDI*, Santa Clara, CA, USA, 2016. (Cited on pages 1 and 6.)

[14] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett. SDX: A Software Defined Internet Exchange. In *ACM SIGCOMM*, Chicago, IL, USA, 2014. (Cited on pages 1 and 6.)

[15] S. Hermans and J. Schutrup. On the Feasibility of Converting AMS-IX to an Industrial-Scale Software Defined Internet Exchange Point, 2016. (Cited on pages 1 and 6.)

[16] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking for Networks. In *USENIX NSDI*, San Jose, CA, USA, 2012. (Cited on pages 2 and 6.)

[17] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying Network-Wide Invariants in Real Time. In *USENIX NSDI*, Lombard, IL, USA, 2013. (Cited on pages 2 and 6.)

[18] R. Klöti, B. Ager, V. Kotronis, G. Nomikos, and X. Dimitropoulos. A Comparative Look into Public IXP Datasets. *ACM SIGCOMM CCR*, 46:21–29, 2016. (Cited on page 5.)

[19] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz. Towards an Accurate AS-level Traceroute Tool. In *ACM SIGCOMM*, Karlsruhe, Germany, 2003. (Cited on page 6.)

[20] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path Splicing. In *ACM SIGCOMM*, Seattle, WA, USA, 2008. (Cited on pages 2 and 6.)

[21] G. Nomikos and X. Dimitropoulos. traixroute: Detecting ixps in traceroute paths. In *PAM*, Heraklion, Greece, 2016. (Cited on page 5.)

[22] SDX Project. SDX-Ryu, 2015. https://github.com/sdn-ixp/sdx-ryu. (Cited on page 5.)

[23] I. Van Beijnum, J. Crowcroft, F. Valera, and M. Bagnulo. Loop-Freeness in Multipath BGP through Propagating the Longest Path. In *IEEE ICC*, Dresden, Germany, 2009. (Cited on page 6.)

[24] M. Zhu, J. Li, Y. Liu, D. Li, and J. Wu. TED: Inter-domain Traffic Engineering via Deflection. In *IEEE/ACM IWQoS*, Hong Kong, May 2014. (Cited on pages 2 and 6.)