An Industrial-Scale Software Defined Internet Exchange Point

Arpit Gupta and Robert MacDavid, *Princeton University;* Rüdiger Birkner, *ETH Zürich;* Marco Canini, *Université catholique de Louvain;* Nick Feamster and Jennifer Rexford, *Princeton University;* Laurent Vanbever, *ETH Zürich*

https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/gupta

This paper is included in the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16).

March 16–18, 2016 • Santa Clara, CA, USA

ISBN 978-1-931971-29-4

Open access to the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16) is sponsored by USENIX.

An Industrial-Scale Software Defined Internet Exchange Point

Arpit Gupta[⋆], Robert MacDavid[⋆], Rüdiger Birkner[†],
Marco Canini[◇], Nick Feamster[⋆], Jennifer Rexford[⋆], Laurent Vanbever[†]
[⋆]Princeton University [†]ETH Zürich [◇]Université catholique de Louvain
http://sdx.cs.princeton.edu/

Abstract

Software-Defined Internet Exchange Points (SDXes) promise to significantly increase the flexibility and function of interdomain traffic delivery on the Internet. Unfortunately, current SDX designs cannot yet achieve the scale required for large Internet exchange points (IXPs), which can host hundreds of participants exchanging traffic for hundreds of thousands of prefixes. Existing platforms are indeed too slow and inefficient to operate at this scale, typically requiring minutes to compile policies and millions of forwarding rules in the data plane.

We motivate, design, and implement iSDX, the first SDX architecture that can operate at the scale of the largest IXPs. We show that iSDX reduces both policy compilation time and forwarding table size by two orders of magnitude compared to current state-of-the-art SDX controllers. Our evaluation against a trace from one of the largest IXPs in the world found that iSDX can compile a realistic set of policies for 500 IXP participants in less than three seconds. Our public release of iSDX, complete with tutorials and documentation, is already spurring early adoption in operational networks.

1 Introduction

Software-Defined Networking (SDN) has reshaped the design of many networks and is poised to enable new capabilities in interdomain traffic delivery. A natural place for this evolution to occur is at Internet exchange points (IXPs), which are becoming increasingly prevalent, particularly in developing regions. Because many Autonomous Systems (ASes) interconnect at IXPs, introducing flexible control at these locations makes it easier for them to control how traffic is exchanged in a direct, and more fine-grained way. In previous work [14], we offered an initial design of a Software-Defined Internet Exchange Point (SDX) and showed how introducing SDN functionality at even a single IXP can catalyze new traffic-management capabilities, ranging from better inbound traffic engineering to application-specific peering and server load balancing.

Since we introduced SDX [13], many organizations and networks have built different versions of this concept [4, 14, 22, 23, 35]. Yet, many of these deployments remain relatively small-scale or limited in scope because current switch hardware cannot support large forwarding tables, and because efficiently combining the policies of independently operated networks as routes and policies change presents a significant scaling challenge.

In this paper, we tackle these scalability challenges with the design and implementation of iSDX, an industrial-scale SDX that can support interconnection for the largest IXPs on the Internet today. We design mechanisms that allow the number of participants, BGP routes, and SDN policies to scale, even for the limited table sizes of today's switches. We develop algorithms for compiling traffic control policies at the scale and speed that networks that such an IXP would require. We have implemented these algorithms in Ryu [31], a widely used SDN controller. We have released our implementation to the public with documentation and tutorials; one large government agency has tested iSDX with hardware switches and is using our controller as the basis for a deployment.

In the design and implementation of iSDX, we address two scalability challenges that are fundamental to *any* SDX design. The first challenge relates to how the control plane combines the policies of individual networks into forwarding entries in the data plane. Compiling traffic control policies expressed in a higher-level policy language to forwarding table entries can be slow, since this process involves composing the policies of multiple participants into a single coherent set of forwarding-table entries. This slow process is exacerbated by the fact that any change to BGP routing may change forwarding behavior; existing SDX designs trigger recompilation every time a BGP best route changes, which is not tractable in practice. The main scalability challenge thus involves efficiently composing the policies of individual participants, and ensuring that the need to recompile the forwarding table entries is completely decoupled from (frequent) BGP routing changes.

To scale the control plane, we introduce a new design that exploits the fact that each participant expresses its SDN policy independently, which implies that each participant can also compile its SDN policies independently, as well. This change enables more aggressive compression of the forwarding tables than is possible when all of the policies are compressed together and also allows for participant policies to be compiled in parallel. As a result, iSDX compiles the forwarding tables two orders of magnitude faster than the existing approaches; the tables are also two orders of magnitude smaller, making them suitable for practical hardware-switch deployments.

The second challenge relates to the data plane: the number of forwarding table entries that might go into the forwarding table at an IXP switch can quickly grow unacceptably large. Part of the challenge results from the fact that the policies that each network writes have to be consistent with the BGP routes that each participant advertises, to ensure that an SDN policy cannot cause the switch to forward traffic on a path that was never advertised in BGP. This process significantly inflates the number of forwarding table entries in the switch and is a considerable deployment hurdle. Large industrial-scale IXPs can have over 700 participants exchanging traffic for hundreds of thousands of prefixes; combined with the fact that each of these participants may now introduce policies for specific traffic flows, the number of forwarding table entries quickly becomes intractable. Although our initial design [14] reduced the size of the forwarding tables, we show that the size of these tables remained prohibitively large for industrial-scale deployments.

To address the data-plane challenge, we introduce an efficient encoding mechanism where the IXP fabric forwards the packet based on an opaque tag that resides in the packet's destination MAC field. This tag explicitly encodes both the next-hop for the packet and the set of ASes that advertise BGP routes for the packet's destination, thus making it possible to remove this information from the switch tables entirely. This separation prevents BGP routing updates from triggering recomputation and recompilation of the forwarding table entries. Using features in OpenFlow 1.3 that support matching on fields with arbitrary bitmasks, we significantly reduce the size of this table by grouping tags with common bitmasks.

In summary, we present the following contributions:

- The design and implementation of iSDX, the first SDX controller that scales to large industrial-scale IXPs. We devised new mechanisms for distributing control-plane computation, compressing the forwarding tables, and responding to BGP routing changes, reducing the compilation time and forwarding table size by several orders of magnitude. (Sections 3–5)
- A public, open-source implementation of iSDX on Github [16]; the system is based on Ryu, a widely used SDN controller, and is accompanied with tutorials and instructions that have already helped spur early adoption. (Section 6)
- An extensive evaluation of iSDX's scalability characteristics using a trace-driven evaluation from one of the largest IXPs in the world. Our evaluation both demonstrates that iSDX can scale to the largest IXPs and provides insight into specifically how (and to what extent) each of our optimizations and algorithms helps iSDX scale. (Section 7)

We survey related work in Section 8 and conclude in Section 9 with a discussion of open issues in SDX design and avenues for future work.

2 SDX: Background & Scaling Challenges

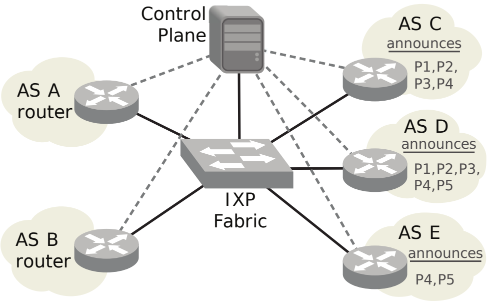
We begin with a background on our previous SDX designs [14, 35] and a demonstration that these designs cannot scale to industrial IXPs.

2.1 Background

Brief overview of SDX. An SDX is an IXP consisting of a programmable SDN fabric, coupled with a BGP route server (which allows IXP participants to exchange reachability information via BGP) and an SDN controller (which allows participants to override default BGP routing behavior with more fine-grained SDN policies). The SDX controller provides each participant AS with the abstraction of a dedicated switch that it can program using match-action policies to control traffic flows. Participants may express SDN policies on both their inbound and outbound traffic; the SDX controller ensures that no SDN policy results in traffic being forwarded to a neighboring AS that did not advertise a BGP route for the prefix that matches the packet's destination IP address.

Each participant runs an SDN control application on the central controller and has its border router exchange BGP update messages with the IXP's route server. The SDN controller combines the SDN policies from all participants, reconciles the resulting policy with the BGP routing information, and computes and installs the resulting forwarding table entries in the IXP fabric. To avoid having forwarding entries for all prefixes, our original SDX design relied on the participants' border routers to tag packets entering the IXP fabric with a *forwarding equivalence class* of destination prefixes with the same forwarding action. For backwards compatibility, the tag was the destination MAC address, set in response to the border router sending an ARP query for the next-hop IP address from the BGP route advertisement. The SDX route server computed a different (virtual) next-hop IP address for each equivalence class of prefixes to trigger the border router to use a common MAC address for packets sent to the group of destination IP addresses.

Example operation. Figure 1a shows an example topology with five participants; Figure 1b shows the routes advertised to A and B and the BGP routes that they select



(a) *Example Topology*

	A	B
P1	C, **D**	**C**, D
P2	C, **D**	**C**, D
P3	C, **D**	**C**, D
P4	C, **D**, E	C, D, **E**
P5	D, **E**	D, **E**

(b) *Reachability and Next Hops (in bold) for AS A and AS B*

Figure 1: *An example with five IXP participants. Two participants AS A and AS B have outbound policies. The other three advertise five IP prefixes to both these participants.*

for each prefix (in bold). Both A and B express outbound policies. To ensure that SDN policies cause the IXP to forward traffic in a way that is consistent with the advertised BGP routes, the SDX controller *augments* each outbound policy with the reachability information. Intuitively, augmentation restricts forwarding policies so that traffic is forwarded only on paths that correspond to BGP routes that the participant has learned.

For example, suppose that A has the following outbound policies:

$\texttt{dPort=443} \rightarrow \texttt{fwd}(C)$

$\texttt{dPort=22} \rightarrow \texttt{fwd}(C)$

$\texttt{dPort=80} \wedge \texttt{sIp=10/24} \rightarrow \texttt{fwd}(D)$

$\texttt{dPort=80} \wedge \texttt{sIp=40/24} \rightarrow \texttt{fwd}(D)$

These policies forward traffic based on values of packet header fields, overriding BGP behavior. For instance, the first policy specifies HTTPS traffic (dPort=443) should be forwarded to C. Without augmentation, A would also forward the HTTPS traffic destined for prefix $P5$ to C, even though C never advertised a path for $P5$ to A. In our example, A's policies are then augmented as follows:

$\texttt{dIp} \in \{\mathbf{P1}, \mathbf{P2}, \mathbf{P3}, \mathbf{P4}\} \wedge \texttt{dPort=443} \rightarrow \texttt{fwd}(C)$

$\texttt{dIp} \in \{\mathbf{P1}, \mathbf{P2}, \mathbf{P3}, \mathbf{P4}\} \wedge \texttt{dPort=22} \rightarrow \texttt{fwd}(C)$

$\texttt{dIp} \in \{\mathbf{P1}, \mathbf{P2}, \mathbf{P3}, \mathbf{P4}, \mathbf{P5}\} \wedge \texttt{dPort=80} \wedge \texttt{sIp=10/24} \rightarrow \texttt{fwd}(D)$

$\texttt{dIp} \in \{\mathbf{P1}, \mathbf{P2}, \mathbf{P3}, \mathbf{P4}, \mathbf{P5}\} \wedge \texttt{dPort=80} \wedge \texttt{sIp=40/24} \rightarrow \texttt{fwd}(D)$

Augmentation enforces that the destination IP (dIp) matches one of the prefixes that either C or D announces to A, therefore ensuring congruence with BGP routing. Observe that a straightforward realization of this policy requires one distinct match-action rule for each of the five prefixes. Hence, the augmented policies would result in 18 forwarding rules instead of the four rules necessary to implement the original policy.

Similarly, if B's outbound policy is:

$\texttt{dPort=443} \rightarrow \texttt{fwd}(E)$

the SDX controller augments the policy, doubling the number of necessary rules, as follows:

$\texttt{dIp} \in \{\mathbf{P4}, \mathbf{P5}\} \wedge \texttt{dPort=443} \rightarrow \texttt{fwd}(E)$

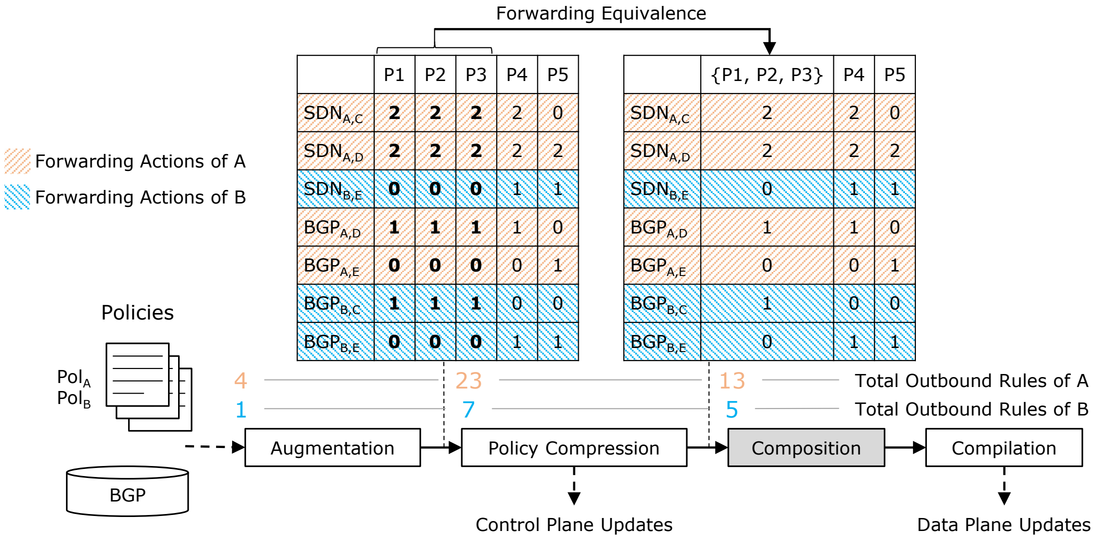
To better illustrate the scalability challenge, we capture the expansion of the switch forwarding tables using an *augmentation matrix* (Figure 2, left matrix). In this matrix, a row labeled as $\mathrm{SDN}_{X,Y}$ refers to an SDN policy written by X that results in traffic being forwarded to Y, while columns refer to IP prefixes. The value of an element (i, j) indicates the number of forwarding table entries (*i.e.*, match-action rules) in participant i's policy where prefix j appears. Similarly, $\mathrm{BGP}_{X,Y}$ indicates whether X selects Y as the next hop for some BGP-advertised prefix, and element (i, j) is 1 if participant A selects the route advertised by B for the prefix corresponding to column j.

For example, the element in row $\mathrm{SDN}_{A,C}$ and column $P1$ reflects the fact there are two forwarding table entries that correspond to prefix $P1$: one for traffic with dPort=443 and one for traffic with dPort=22. The same applies for columns $P2$, $P3$, and $P4$. We can determine the total number of forwarding table entries (and the number contributed by each participant) by summing up the corresponding elements in the matrix. We will use this notation to describe compression techniques (and their effects) throughout the paper.

Previously developed compression techniques. Intuitively, the number of forwarding rules increases as the number of SDX participants with outbound policies increases (more rows) and as forwarding policies are defined for additional prefixes (more columns). To limit the number of forwarding rules, the original SDX design [14] identified the Minimum Disjoint Set (MDS) of prefixes (columns) with the same SDN policies and grouped each equivalent set into a Forwarding Equivalence Class (FEC). In the rest of this paper, we refer to this algorithm as *MDS compression*. For instance, in the preceding example, prefixes $P1, P2, P3$ belong to the same FEC, as indicated by the boldface entries in the left matrix in Figure 2. MDS compression reduces the number of forwarding table entries by assigning a virtual next-hop to each FEC, rather than to each individual prefix. Figure 2 also depicts the number of forwarding table entries before and after MDS compression. In particular, MDS compression reduces the number of columns from the total number of prefixes (5) to the number of FECs (3).

2.2 Existing SDX Designs Do Not Scale

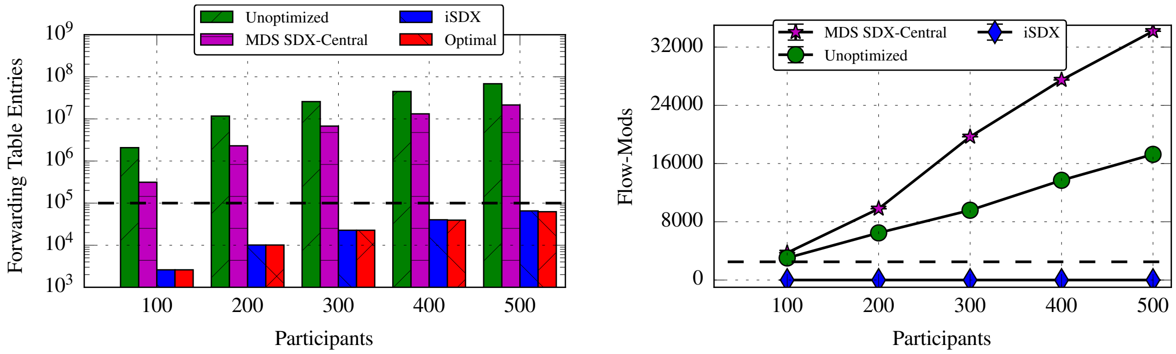
In this section, we show that existing SDX designs do not scale to the demands of industrial-scale IXPs. We explore two different state-of-the-art SDX designs: (1) an



	P1	P2	P3	P4	P5
$SDN_{A,C}$	**2**	**2**	**2**	2	0
$SDN_{A,D}$	**2**	**2**	**2**	2	2
$SDN_{B,E}$	**0**	**0**	**0**	1	1
$BGP_{A,D}$	**1**	**1**	**1**	1	0
$BGP_{A,E}$	**0**	**0**	**0**	0	1
$BGP_{B,C}$	**1**	**1**	**1**	0	0
$BGP_{B,E}$	**0**	**0**	**0**	1	1

	{P1, P2, P3}	P4	P5
$SDN_{A,C}$	2	2	0
$SDN_{A,D}$	2	2	2
$SDN_{B,E}$	0	1	1
$BGP_{A,D}$	1	1	0
$BGP_{A,E}$	0	0	1
$BGP_{B,C}$	1	0	0
$BGP_{B,E}$	0	1	1

Figure 2: *Matrix representation of AS A and AS B's outbound policies after augmentation and policy compression, as well as the stages of compression and composition in the original SDX design; the composition stage is grey to indicate that the iSDX eliminates this stage entirely.*



(a) *Number of Forwarding Table Entries.*

(b) *Data-Plane Update Rate.*

Figure 3: *Existing SDX designs can require to maintain millions of forwarding entries (left) and update 10,000s of updates per second (right). Such numbers are far from current hardware capabilities. As an illustration, the dashed line highlights the hardware capabilities of state-of-the-art SDN switches [26].*

unoptimized SDX that does not compress policies, such as that used by Google's Cardigan SDX [38]; (2) a simple, centralized SDX controller that applies MDS compression, as in our previous work [14]. We also preview the results from this paper, showing that our new architecture, iSDX, reduces the compilation time, number of forwarding table entries, and data-plane update rate by more than two orders of magnitude, thus making operation in an industrial-scale IXP practical. In each case, we evaluate the time to compute the forwarding table entries, the number of forwarding table entries, and the rate at which changes in BGP routing information induce changes in the forwarding table entries. We use a real BGP trace from one the largest IXPs in the world for this evaluation. Section 7 provides details about our experiment setup.

	Unoptimized	Centralized MDS-SDX [14]	iSDX
Time (s)	4572.15	1740.93	2.82

Table 1: *Median time (for 60 trials) to compute forwarding table entries for an IXP with 500 participants. The iSDX column shows the results for this paper.*

Existing SDX designs can take minutes to compute forwarding table entries. Table 1 shows the median time over 60 trials to compute forwarding table entries for an IXP with 500 participants for two state-of-the-art SDX designs, as well as for iSDX, the design that we present in this paper. iSDX reduces the average time to compute forwarding table entries from 30 minutes to *less than three seconds.*

Existing SDX designs can require millions of forwarding table entries. Figure 3a shows how the number of forwarding table entries increases as the number of participants increases from 100 to 500. MDS compression reduces the number of entries by an order of magnitude, but the forwarding table is still too large for even the most high-end hardware switches, which have about 100,000 TCAM entries [26]. The iSDX design ensures that the number of forwarding table entries is approximately the number of SDN policies that each participant expresses (shown as "optimal" in Figure 3a), thus allowing the number of forwarding table entries to be in the tens of thousands, rather than tens of millions.

Existing SDX designs require hundreds of thousands updates per second to the data plane. Figure 3b shows the worst-case data-plane update rate that an SDX controller must sustain to remain consistent with existing BGP updates. The update rates of existing designs are several orders of magnitude above what even top-of-the-line hardware switches can support [26] (*i.e.*, about 2,500 updates per second). In constrast, iSDX usually eliminates forwarding table updates in response to BGP updates.

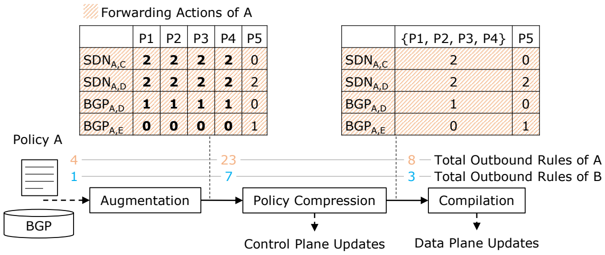
3 Design of an Industrial-Scale SDX

We introduce the design of an industrial-scale SDX (iSDX), which relies on two principles to reduce compilation time, the number of forwarding table entries, and forwarding table update rates.

3.1 Partition Control-Plane Computation

Problem: Considering all policies together reduces opportunities for compression. Centralized SDX controllers perform control-plane computations for all IXP participants. Doing so not only forces the controller to process a large single combined policy, it also creates dependencies between the policies of individual IXP participants. For example, a change to any participant's inbound policy triggers the recompilation of the policies of *all* participants who forward traffic to that participant. This process requires significant computation and also involves many (and frequent) updates to the forwarding table entries at the IXP switch.

Solution: Partition computation across participants. We solve this problem by partitioning the control-plane computation across participants. Doing so ensures that participant policies stay independent from each other. In addition, partitioning the computation enables more efficient policy compression by operating on smaller state, reducing both computation time and data plane state. Partitioning the control-plane computation among participants also enables policy compilation to scale out as the number of IXP participants and routes grows. Section 4 details this approach.



	P1	P2	P3	P4	P5
$SDN_{A,C}$	**2**	**2**	**2**	**2**	0
$SDN_{A,D}$	**2**	**2**	**2**	**2**	2
$BGP_{A,D}$	**1**	**1**	**1**	**1**	0
$BGP_{A,E}$	**0**	**0**	**0**	**0**	1

	{P1, P2, P3, P4}	P5
$SDN_{A,C}$	2	0
$SDN_{A,D}$	2	2
$BGP_{A,D}$	1	0
$BGP_{A,E}$	0	1

Figure 4: *Partitioning the Control-Plane Computation.*

3.2 Decouple BGP and SDN Forwarding

Problem: Frequent BGP updates trigger recompilation. Coupling BGP and SDN policies during compilation inflates the number of resulting forwarding table entries and also implies that any change to BGP routing triggers recompilation of the forwarding table entries, which is costly. Our previous design partially addressed this problem, but this design still requires millions of flow rules in the data plane as shown in Figure 3a. Additionally, our previous approach to reduce the number of forwarding table entries *increases* the forwarding table update rates, since any change in BGP routing may affect how entries are compressed.

Solution: Encode BGP reachability information in a separate tag. We address this problem by encoding all information about BGP best routes (and corresponding next hops) into the destination MAC addresses, which reduces the number of forwarding table entries, as well as the number of changes to the forwarding table when BGP routes change. Section 5 discusses our approach in detail.

4 Partitioning Control-Plane Computation

To achieve greater compression of the rule matrix, we need to reduce the constraints that determine which prefixes belong to the same FEC. Rather than computing one set of equivalence classes for the entire SDX, iSDX computes *separate FECs for each participant*. We first discuss how partitioning by participant reduces the size of the rule matrices and, as a side benefit, allows for faster computation. We then describe how we use multiple match-action tables and ARP relays to further improve scalability, setting the stage for further optimizations in Section 5.

4.1 Partitioning the FEC Computation

Figure 4 shows similar compression and compilation steps as the ones done in Figure 2, with the important distinction that it takes place on behalf of participant *A* only; similar operations take place on behalf of other participants. Fig-

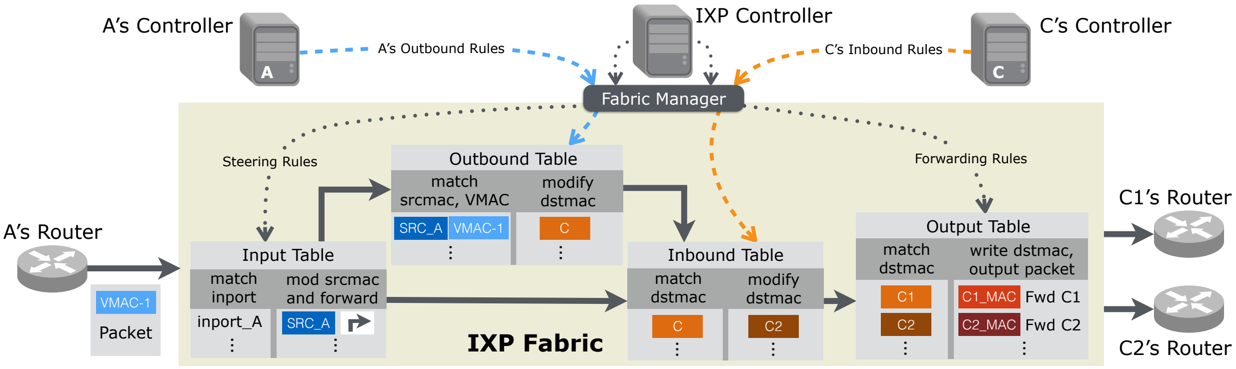


Figure 5: *Distributing forwarding rules and tags.*

ure 4 highlights two important benefits of partitioning the computation of FEC across participants:

- Computing separately for each participant reduces the number of next-hops, leading to a smaller number of larger forwarding equivalence classes. In Figure 4, the number of columns reduces from five to two.
- The computational complexity of computing FECs is proportional to the number of rows times the number of columns in the rule matrix. Now, each rule matrix is smaller, and the computation for different participants can be performed in parallel.

In practice, the SDX controller could compute the FECs for each participant, or each participant could run its own controller for computing its own FECs. In the rest of the paper, we assume each participant runs its own controller for computing its FECs.

4.2 Distributing Forwarding Rules & Tags

In addition to computing the FECs for each participant, the iSDX must realize these policies in the data plane.

Decomposing the IXP fabric into four tables: To forward traffic correctly, an SDX must combine the inbound and outbound policies for all of the participants. Representing the combination of policies in a single forwarding table, as in an OpenFlow 1.0 switch, would be extremely expensive. Some existing SDN controllers perform this type of composition [25, 33]—essentially computing a cross product of the constituent policies—and, in fact, our original SDX followed this approach [14]. Computing the cross product leads to an explosion in the number of rules, and significant recomputation whenever one of the participant policies changes.

Fortunately, modern switches have multiple stages of match-action tables, and modern IXPs consist of multiple switches. The iSDX design capitalizes on this trend. The main challenge is to determine *how* to most effectively map policies to the underlying tables.

A strawman solution would be to use a two-table pipeline, where packets first enter an outbound table implementing outbound policies for the participant where the traffic originates, followed by an inbound table that applies inbound policies for the participant that receives the traffic as it leaves the IXP fabric. Using only two tables, however, would mean that some of these tables would need to be much larger; for example, the outbound table would need to represent the cross product of all input ports and outbound policies. Additionally, using only two tables makes it more difficult to scale-out the iSDX as the number of participants grows.

As such, our design incorporates an input table, which handles all the incoming traffic and tags it with a new source MAC address based on the packet's incoming port, so that packets can be multiplexed to the outbound table. As the packet leaves the iSDX, it passes through an output table, which looks up the packet's tag in the destination MAC field and both performs the appropriate action and rewrites the packet's destination MAC address. Separate input and output tables provide a cleaner separation of function between the modules that write to each table, avoid cross-product explosion of policies, and facilitates scale-out by allowing the inbound and outbound tables to reside on multiple physical switches in the IXP infrastructure. (Such scale-out techniques are beyond of the scope of this paper.)

Figure 5 shows how the IXP fabric forwards a packet, while distributing the compilation and compression of policies across separate tables. Based on the destination IP address of the packet, suppose that AS *A*'s controller selects a route to the packet's destination via AS *D*; this route will correspond to a next-hop IP address. AS *A*'s controller will make a BGP announcement advertising this path. AS *A*'s router will issue an ARP query for the advertised next-hop IP address, and then AS *A*'s controller will respond via the ARP relay setting a virtual MAC address (in Figure 5, "VMAC-1") as the packet's destination MAC address.

When the packet enters the IXP fabric, the input table matches on the packet's incoming port and rewrites the source MAC address to indicate that the packet arrived from AS A (“SRC_A”). If A has an outbound policy, the packet will match on (“SRC_A”), and the outbound table will apply an outbound policy. If A has no outbound policy for this packet, the input table forwards the packet directly to the inbound table without changing the destination MAC. This bypass is not strictly necessary but avoids an additional lookup for packets that do not have a corresponding outbound policy. A’s outbound policy thus overwrites default BGP forwarding decision and modifies the destination MAC address to “C”. The inbound table rewrites the tag to correspond to the final disposition of the packet (“C1” or “C2”), which is implemented in the output table. The output table also rewrites the tag to the receiver’s physical MAC address before forwarding.

Reducing ARP traffic overhead. Partitioning the FEC computation reduces the number of FECs per participant, but may increase the *total* number of FECs across all participants (*i.e.*, the number of columns across all rule matrices). To reduce the size of the forwarding tables, each data packet carries a tag (*i.e.*, a virtual MAC address) that identifies its FEC. The participant’s border router learns the virtual MAC address through an ARP query on the BGP next-hop IP address of the associated routes. The use of broadcast for ARP traffic, combined with the larger number of next-hop IP addresses, could overwhelm the border routers and the IXP fabric. In fact, today’s IXPs are already vulnerable to high ARP overheads [5].

Fortunately, we can easily reduce the overhead of ARP queries and responses, because each participant needs to learn about only the virtual MAC addresses for its own FECs. As such, the SDX can turn ARP traffic into *unicast* traffic by installing the appropriate rules for handling ARP traffic in switches. In particular, each participant’s controller broadcasts a gratuitous ARP response for every virtual next-hop IP address it uses; rules in the IXP’s fabric recognize the gratuitous ARP broadcasts and ensure that they are forwarded only to the relevant participant’s routers. Participants’ routers can still issue ARP queries to map IP addresses to virtual MAC addresses, but the fabric intercepts these queries and redirects them to an ARP relay to avoid overwhelming other routers.

5 Decoupling SDN Policies from Routing

To ensure correctness, any SDX platform must combine SDN policies with dynamic BGP state: which participants have routes to each prefix (*i.e.*, valid next-hop ASes for a packet with a given destination prefix), as well as the next-hop AS to use for each prefix (*i.e.*, the outcome of BGP decision process). The large number of prefixes and participants creates scalability challenges with respect to forwarding table sizes and update rates, before SDN policies even enter the equation.

5.1 Idea: Statically Encode Routing

To reduce the number of rules and updates, we develop a new encoding scheme that is analogous to source routing: The IXP fabric matches on a tag that is provisioned by a participant’s SDX controller. To implement this approach, we optimize the tag that the fabric uses to forward traffic (as described in Section 4) to carry information about both the next-hop AS for the packet (as determined by the best BGP route) and the ASes who have advertised routes to the packet’s destination prefix. If no SDN policy matches a packet, iSDX can simply match on the next-hop AS bits of the tag to make a default forwarding decision. As before, the sender discovers this tag via ARP.

To implement default forwarding, the IXP fabric maintains static entries for each next-hop AS which forward to participants based upon the next-hop AS bits of the tag. When the best BGP routes change, the entries need not change, rather the next-hop AS bits of the tags change.

To account for changes in available routes, SDN policies that reroute to some participant X confirm whether X has advertised a route before forwarding. The method of checking for X in the tags is static, meaning that in contrast to our previous design [14], BGP updates induce zero updates in the IXP switch data plane. Instead, BGP updates result in tag changes, and the participant’s border router learns these dynamic tags via ARP.

5.2 Encoding Next-Hop and Reachability

We now describe how iSDX embeds both the next-hop AS (*i.e.*, from the best BGP route) and the reachability information (*i.e.*, the set of ASes that advertise routes to some prefix) into this tag.

5.2.1 Next-hop encoding

The next-hop information denotes the default next-hop AS for a packet, as determined by BGP. In the example from Section 2.1, A’s next-hop AS for traffic to $P1$ as determined by the best BGP route is D. iSDX allocates bits from the tag (*i.e.*, the virtual MAC, which is written into the destination MAC of the packet’s header) to denote this next-hop. If no SDN policy overrides this default, iSDX applies a default priority prefix-based match on these bits to direct traffic to the corresponding next-hop.[1] This approach reduces the forwarding table entries in a participant’s outbound table, since additional entries for default BGP forwarding no longer need to be represented as distinct entries in the forwarding table. Encoding the next hop information in this way requires $\lg(N)$ bits, where N is the number of IXP participants. At a large IXP with up to 1024 participants, ten bits can encode information about default next-hop ASes, leaving 37 bits.[2]

[1] The OpenFlow 1.3 standard supports this feature [27], which is already implemented in many hardware switches (*e.g.*, [26, 28]).

[2] One of the 48 bits in the MAC header is reserved for multicast.

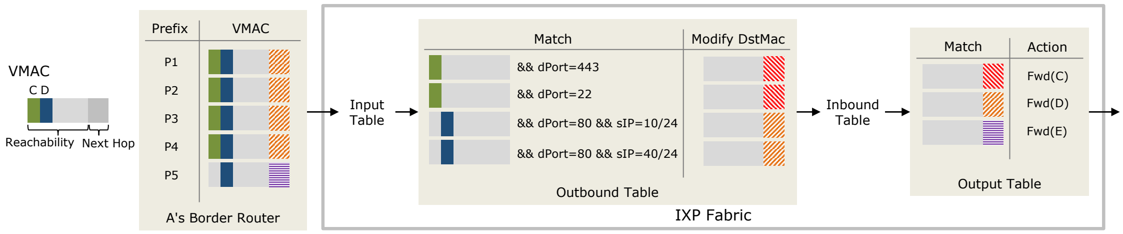


Figure 6: *How AS A's controller uses reachability encoding to reduce the number of flow rules.*

5.2.2 Reachability encoding

We now explain how to encode reachability information into the remaining 37 bits of the destination MAC address. We first present a strawman approach that illustrates the intuition before describing the scalable encoding.

Strawman encoding. Suppose that for a given tag, the i-th bit is 1 if that participant learns a BGP route to the corresponding prefix (or prefixes) via next-hop AS i. Such an encoding would allow the IXP fabric to efficiently determine whether some participant could forward traffic to some next-hop AS i, for any i at the IXP. Considering the example in Section 2.1, A's outbound policies are:

dMac = XX**1**X...X ∧ dPort=443 → fwd(C)
dMac = XX**1**X...X ∧ dPort=22 → fwd(C)
dMac = XXX**1**X..X ∧ dPort=80 ∧ sIp=10/24 → fwd(D)
dMac = XXX**1**X..X ∧ dPort=80 ∧ sIp=40/24 → fwd(D)

where X stands for a wildcard match (0 or 1). This encoding ensures correct interoperation with BGP, yet we use just four forwarding table entries, which is fewer than the 18 required using augmentation (from the original example in Section 2).

Figure 6 explains how this approach reduces the number of forwarding table entries in the switch fabric. When a packet arrives, its virtual MAC encodes both (1) which ASes have advertised a BGP route for the packet's destination ("reachability") and (2) the next-hop participant corresponding to the best BGP route ("next hop"). Suppose that a packet is destined for $P1$ from A; in this case, A's border router will affix the virtual MAC as shown. If that virtual MAC does not match any forwarding table entries in the outbound table, the packet will simply be forwarded to the appropriate default next hop (in this case, D) based on the next-hop encoding. This process makes it possible for the switch to forward default BGP traffic without installing any rules in the outbound table, significantly reducing the size of this table.

Hierarchical encoding. The approach consumes one bit per IXP participant, allowing at most for only 37 IXP participants. To encode more participant ASes in these 37 bits, we divide this bitspace hierarchically. Suppose that an IXP participant has SDN policies that refer to N other IXP participants (*i.e.*, possible next-hop ASes). Then, all of these N participants need to be efficiently encoded in the 37-bit space, B. We aim to create W bitmasks $\{B_1, B_2, \ldots, B_W\}$ that minimize the total number of forwarding table entries, subject to the limitations of the total length of the bitmask.

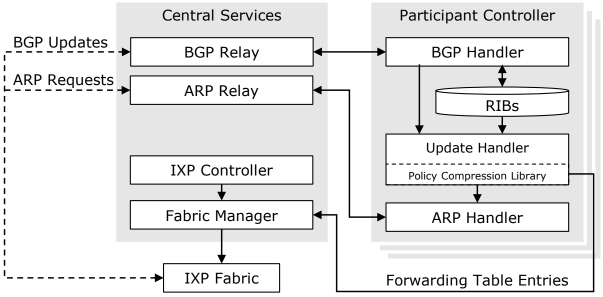


Figure 7: *Implementation of iSDX. It has five main modules: (1) IXP controller, (2) participant SDN controller, (3) ARP relay, (4) BGP relay, and (5) fabric manager.*

Given M prefixes and N IXP participants, we begin with M bitmasks, where each bitmask encodes some *set* of participants that advertise routes to some prefix p_i. We greedily merge pairs of sets that have at least one common participant, and we always merge two sets if one is a subset of the other. Iterating over all feasible merges has worst-case complexity $O(M^2)$; and there may be as many as $M-1$ merge actions in the worst case. Each merge has complexity $O(N)$, which gives us an overall worst-case running time complexity of $O(M^3N)$.

Given 37 spare bits in the destination MAC for reachability encoding, if a participant has defined SDN policies for more that 37 participants who advertise the same prefix, then the number of bits required to encode the reachability information will exceed 37. Our analysis using a dataset from one of the largest IXPs in the world found that the maximum number of participants advertising the same prefix was only 27, implying that largest bitmask that this encoding scheme would require is 27 bits. There were 62 total bitmasks, meaning 6 bits are required to encode the ID of a bitmask, requiring a total of 33 bits for the encoding. Using a different (or custom) field in a packet header might also be possible if these numbers grow in the future.

6 Implementation

We now describe an implementation of iSDX, as shown in Figure 7. Our Python-based implementation has about 5,000 lines of code. Source code and tutorials are publicly available on Github [16]. We have also provided instructions for deploying iSDX on hardware switches [17]; one large government agency has successfully done so with the Quanta LY2 switch [28]. About 300 students used an earlier version of iSDX in the Coursera SDN course [8].

The **fabric manager** is based on Ryu [31]. It listens for forwarding table modification instructions from the participant controllers and the IXP controller and installs the changes in the switch fabric. The fabric manager abstracts the details of the underlying switch hardware and OpenFlow messages from the participant and the IXP controllers and also ensures isolation between participants.

The **IXP controller** installs forwarding table entries in the input and output tables in the switch fabric via the fabric manager. Because all of these rules are static, they are computed only at initialization. Moreover, the IXP controller handles ARP queries and replies in the fabric and ensures that these messages are forwarded to the respective participants' controllers via **ARP relay**.

The **BGP relay** is based on ExaBGP [11] and is similar to a BGP route server in terms of establishing peering sessions with the border routers. Unlike a route server, it does not perform any route selection. Instead, it multiplexes all BGP routes to the participant controllers.

Each **participant SDN controller** computes a compressed set of forwarding table entries, which are installed into the inbound and outbound tables via the fabric manager, and continuously updates the entries in response to the changes in SDN policies and BGP updates. The participant controller receives BGP updates from the BGP relay. It processes the incoming BGP updates by selecting the best route and updating the RIBs. We developed APIs to use either of MongoDB [24], Cassandra [2] and SQLite [34] for storing participants' RIBs. We used the MongoDB (in-memory) for the evaluation in Section 7. The participant controller also generates BGP announcements destined to the border routers of this participant, which are sent to the routers via the BGP relay.

Each participant controller's **update handler** determines whether the inbound and outbound tables need to be updated, as well as whether new gratuitous ARP messages must be sent to the participant's border routers to update any virtual destination MAC addresses. The controller receives ARP requests from the participant's border routers via the **ARP handler** and determines the corresponding ARP reply. The controller also receives SDN policy updates from the network operators in the form of addition and removal lists. Both the update handler and the ARP handler use a policy compression library that we implemented, which provides the mapping between IP prefixes and virtual next-hop IPs (corresponding to best BGP routes), and between virtual next-hop IPs and virtual destination MAC addresses (*i.e.*, an ARP table).

	MDS	NH Encoding	Reachability Encoding
iSDX-D	✓	✗	✗
iSDX-N	✓	✓	✗
iSDX-R	✗	✓	✓

Table 2: *Three distributed SDX Controllers.*

7 Evaluation

We now demonstrate that iSDX can scale to the forwarding table size, data plane update rate, and control plane computation requirements of an industrial-scale IXP. Table 2 summarizes the three different iSDX designs that we compare to previous approaches: iSDX-D applies the same MDS compression technique as in our previous work [14], but with tables distributed across participants; iSDX-N additionally encodes the next-hop AS in the tag; and iSDX-R encodes both the next-hop AS and BGP reachability information in the tag.

Table 3 summarizes our results: *iSDX reduces the number of forwarding table entries for an industrial-scale IXP by three orders of magnitude as compared to an unoptimized, centralized SDX design; and by more than two orders of magnitude over the state-of-the-art SDX design [14].* This section explains these results in detail.

7.1 Experiment Setup

We use data sets from one of the largest IXPs worldwide, which interconnects more than 600 participants, who peer with one another via a BGP route server [29]. We had access to the RIB table dump collected from the IXP's route server on August 8, 2015 for 511 IXP participants. These datasets contain a total of 96.6 million peering (*i.e.*, non-transit) routes for over 300,000 distinct prefixes. We also use a trace of 25,676 BGP update messages from these participants to the route server for the two hours following the collection of this RIB table dump (the participants' RIBs are naturally not perfectly aligned, since dumping a BGP table of about 36 GB from the router takes about fifteen minutes). Our data set does not contain any user data or any personal information that identifies individual users. We run our experiments on a server installed at this IXP configured with 16 physical cores at 3.4 GHz and 128 GB of RAM.

This IXP does not use a programmable IXP fabric, so we assume how participants *might* specify SDN policies, as described in Section 2.1. Specifically, *each participant* has between one to four outbound policies for each of 10% of the total participants. The number of policies and set of participants are chosen uniformly at random. Our sensitivity analysis on this percentage shows that our

			iSDX		
	Unoptimized	Centralized MDS-SDX [14]	iSDX-D	iSDX-N	iSDX-R
Number of Forwarding Table Entries	68,476,528	21,439,540	763,000	155,000	65,250
Policy Compression Time (s)	N/A	297.493	0.0629	0.111	2.810

Table 3: *Summary of evaluation results for iSDX with 500 IXP participants. Note that compression times for iSDX are per-participant, since each participant can compile policies in parallel; even normalizing by this parallelization still yields significant gains.*

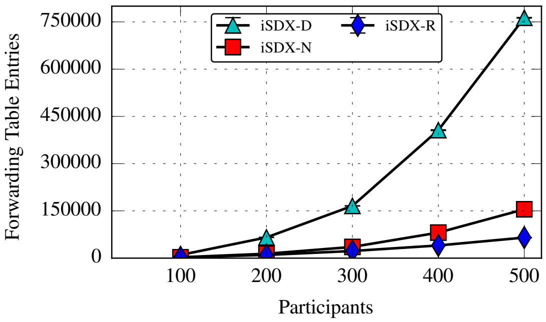


Figure 8: *Number of forwarding table entries.*

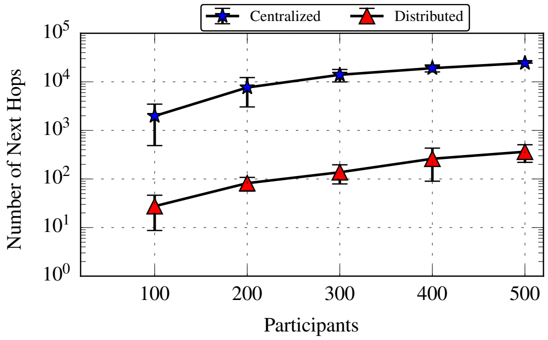


Figure 9: *Number of virtual next-hop IP addresses for centralized and distributed control planes. Results for distributed iSDX do not depend on encoding or compression approach.*

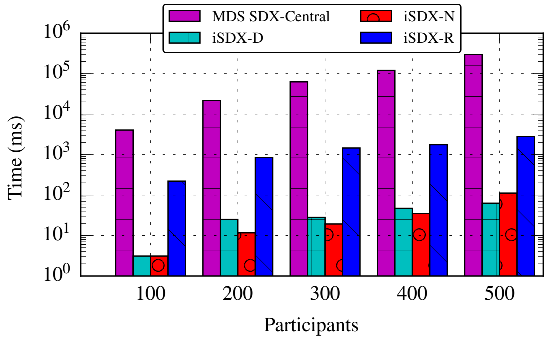


Figure 10: *Time to perform policy compression.*

results are influenced in magnitude but the underlying trends remain. Note that this setup is more taxing than the one in our previous work [14] where only 20% of the total participants had any SDN policies at all. We also evaluate iSDX's performance for smaller IXPs by selecting random subsets of IXP participants (ranging from 100 to 500 ASes) and considering only the RIB information and BGP updates for those participants. We also repeated experiments using public RIB dumps and BGP updates collected by RIPE's RIS servers from 12 other IXPs [30]. As the observed workload was much smaller in this case, we omit these results for brevity.

7.2 Steady-State Performance

We first evaluate the steady-state performance of iSDX. To do so, we use the RIB dumps to initialize the SDX controller (multiple of them for the distributed case) and evaluate the overall performance in terms of the efficiency of data-plane compression, and the time to compile policies and compress them into smaller forwarding tables.

Efficiency of compression. Figure 8 shows the number of forwarding table entries for the three distributed controllers: iSDX-D, iSDX-N, and iSDX-R. The number of forwarding table entries increases with the increasing number of IXP participants. Each of our techniques progressively improves scalability. We observe that the number of forwarding table entries for iSDX-R is very close to the lower bound (*i.e.*, best case), where the number of forwarding table entries is equal to the number of SDN policies.

We also explore the effects of distributing the control plane computation on the ability of iSDX to perform MDS compression. The results are shown in Figure 9. Given 500 participants, partitioning the control plane reduces the number of next hop entries for the border router from 25,000 to 360. This reduction mitigates the load on the border routers, since the number of virtual next hop IP addresses reflects the number of ARP entries each participant's border router must maintain.

Time to perform policy compression. Figure 10 shows the compression time for each controller; this time dominates control-plane computation but only occurs at initialization. The Centralized MDS-SDX operates on a large input rule matrix, and thus requires nearly five minutes to compress policies. iSDX-D distributes the computation across participants, reducing compression time by three orders of magnitude. iSDX-R takes longer than iSDX-D and iSDX-N controllers. For 500 participants, policy compression takes about three seconds.

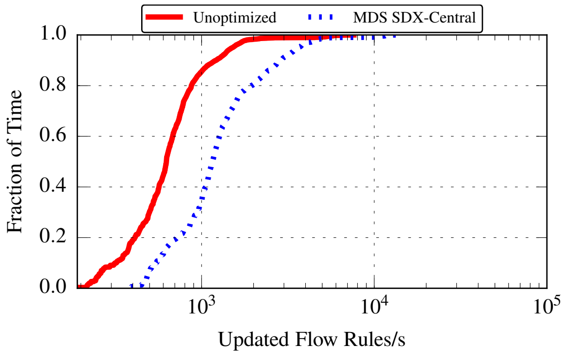


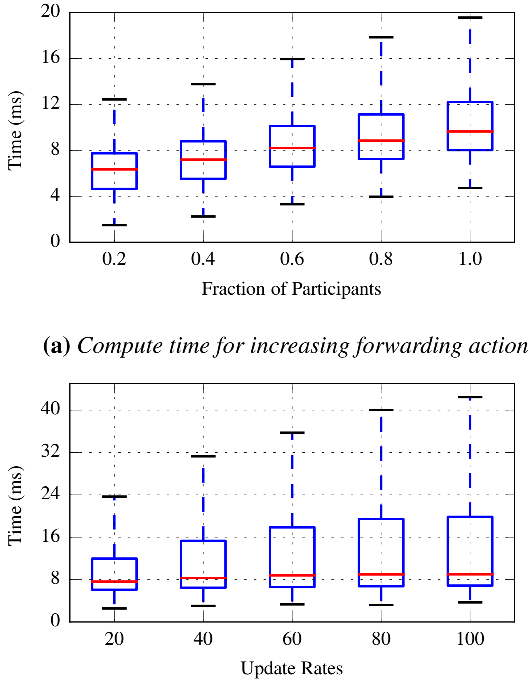
Figure 11: *Rate at which forwarding table entries are updated.*

7.3 Runtime Performance

After iSDX initializes, we replay a two-hour trace of BGP updates from one of the largest IXPs in the world to evaluate the runtime performance of iSDX compared to other SDX designs. We focus on how iSDX reduces the number of forwarding table updates induced by BGP updates and policy changes, as well as the corresponding increase in gratuitous ARP traffic, which is the cost we pay for increased forwarding table stability.

Forwarding table updates in response to routing. Figure 11 shows the cumulative distribution of the number of updated forwarding table entries per second the SDX must process for a BGP update stream coming from all 511 participants at the IXP. MDS compression, which is used in iSDX-D and iSDX-N, significantly increases the rate of updates to the forwarding table in comparison to an unoptimized SDX; this result makes sense because any change to forwarding is more likely to trigger a change to one of the encoded forwarding table entries. *With iSDX-R, there are never updates to the forwarding table entries in response to BGP updates.*

Update latency in response to BGP updates. We aim to understand how quickly iSDX-R can update forwarding information when BGP updates arrive. For iSDX-R, this update time effectively amounts to computing updated virtual next-hop IP and MAC addresses, since iSDX-R never needs to update the IXP fabric forwarding table entries in response to BGP updates. We evaluate update latency with two experiments. First, we vary the fraction of IXP participants to which each IXP participant forwards with SDN policies. For example, if the fraction is 1, each participant has between one and four SDN forwarding policies (at random) for every other SDN participant. Figure 12a shows this result; in all cases, the median update latency in response to a BGP update is less than 10 ms, and the 95th percentile in the worst case is less than 20 ms. Even when we perform simultaneous compilation of all 511 participants on just three servers at the IXP, the median update time is only 52 ms, well within practical requirements.



(a) *Compute time for increasing forwarding actions.*

(b) *Compute time for increasing sustained rates of BGP updates.*

Figure 12: *Latency of iSDX-R updates in response to BGP update streams.*

To understand how iSDX-R behaves when it receives larger update bursts, we evaluate the update latency for increasing sizes of BGP update bursts. We vary the number of BGP updates per second from 20 to 100 and send a constant stream of updates at this rate for five minutes, tracking the latency that the iSDX requires to process the updates. (Although a table reset would presumably cause a very large update burst, the fastest sustained BGP update rate we observed in the trace was only about 35 BGP updates per second.) Figure 12b shows this result. For example, for a rate of 100 BGP updates per second, the median update latency is about 8 ms and the 95th percentile is percentile is about 45 ms.

Gratuitous ARP overhead. Recall that SDX relies on gratuitous ARP to update virtual destination MAC addresses when forwarding behavior changes, often in lieu of updating the forwarding table itself. A centralized SDX control plane sends this ARP response to all IXP participants, but a distributed SDX can send this response only to the border router whose route changed. Figure 13 shows the distribution of the rate at which a participant's border router receives gratuitous ARP messages from the IXP controller in response to BGP routing changes, for both the centralized design (*i.e.*, centralized MDS) and the

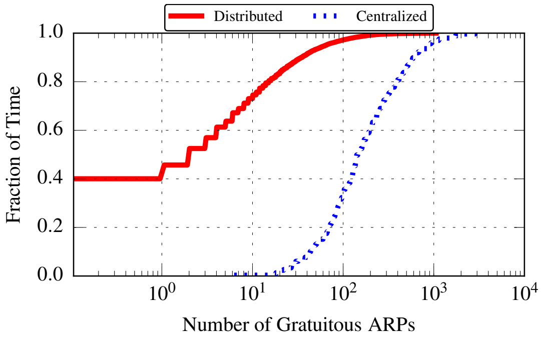


Figure 13: *Rate at which a participant's border router receives gratuitous ARPs.*

distributed one (*i.e.*, iSDX); these rates are independent of which encoding the iSDX uses.

8 Related Work

Ongoing SDX Projects. Software-defined IXPs have been gaining momentum in the past few years [3, 22, 39], and limited real-world deployments are beginning to emerge. Yet, these existing deployments have focused on either smaller IXPs or on forwarding traffic for a partial routing table. Our original SDX [14] work introduced mechanisms for applying SDN policies to control interdomain traffic flow at an IXP and introduced some simple mechanisms for forwarding table compression; yet its capability for compressing and updating forwarding tables cannot meet either the scale or speed demands of the largest industrial IXP. Google's Cardigan SDX controller has been deployed in a live Internet exchange in New Zealand [4, 35]. Cardigan does not use any of the compression techniques that we use in either SDX or iSDX. As a result, we expect that the size of Cardigan's forwarding tables would be similar to the "unoptimized" results that we present in Section 2—orders of magnitude too large for use with hardware switches in large IXPs. Control Exchange Points [21] propose to interconnect multiple SDN IXPs to provide QoS services to the participants and is less concerned with the design of an individual SDN-based IXP.

Distributed SDN controllers. HyperFlow [37], Onix [20], and Devolved Controllers [36] implement distributed SDN controllers that maintain eventually consistent network state partitioning computation across multiple controllers such that each operates on less state. Kandoo [15] distributes the control plane for scalability, processing frequent events in highly replicated local control applications and rare events in a central location. Several distributed controllers focus on fault-tolerance [6, 10, 19]. In contrast to these systems, each participant controller in iSDX operates independently and requires no state synchronization. iSDX's partitioning is first and foremost intended to achieve more efficient compression of forwarding table entries; other benefits, such as parallel computation and fault tolerance, are incidental benefits.

Techniques for data-plane scalability. Other work seeks to address the problem of small forwarding tables in hardware. Data-plane scaling involves (1) rule partitioning [40], where data plane rules are partitioned across multiple switches and incoming traffic is steered to load balance across these switches; and (2) caching [18, 32], which stores forwarding table entries for only a small number of flows in the data plane. These techniques are orthogonal to the compression that iSDX uses. Labeling packets for FIB compression has been applied in various contexts, such as MPLS [9], Fabric [7], LISP [12], and Shadow Macs [1]. These techniques all reduce the number of forwarding table entries in certain routers, often by pushing complex policies to the edge of the network. These techniques generally apply in the wide area, and cannot be directly applied to an IXP topology, although some of the techniques are analogous.

9 Conclusion

Software-Defined Internet Exchange Points (SDXes) are poised to reshape interdomain traffic delivery on the Internet, yet realizing this vision ultimately requires the design and implementation of an SDX that can scale to (and beyond) the largest industrial IXPs on the Internet today. To address this challenge, we developed iSDX, the first SDX controller that scales to large industrial IXPs. We demonstrated how the principles of modularity and decoupling are necessary to scale the control and the data planes. The specific approaches we suggest—partitioning and compression—are applicable in various settings where where composition of forwarding policies is required (*e.g.*, SDN WAN). We have released a public, open-source implementation of iSDX on Github [16], along with tutorials and instructions that have helped catalyze early adoption. Our evaluation shows that iSDX reduces both forwarding table size and the time to compute these entries by several orders of magnitude—enough to make iSDX practical for real operation. Using BGP routing updates from a route server at one of the world's largest IXPs, we showed that iSDX can support industrial-scale operation.

Acknowledgments. We thank our shepherd Dave Oran, Bryan Larish, Jamie Brown, Inder Monga, Rick Porter, Glenn Gardner, Marc Pucci, David Jorm, and the anonymous reviewers for the feedback and comments. This research was supported by National Science Foundation Awards CNS-1539920, CNS-1409056, and a research contract with the Laboratory for Telecommunications Sciences. This research was also supported by European Union's Horizon 2020 research and innovation program under the ENDEAVOUR project (grant agreement 644960).

References

[1] AGARWAL, K., DIXON, C., ROZNER, E., AND CARTER, J. Shadow macs: Scalable label-switching for commodity ethernet. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (New York, NY, USA, 2014), HotSDN '14, ACM, pp. 157–162. (Cited on page 12.)

[2] APACHE CASSANDRA. http://cassandra.apache.org/. (Cited on page 9.)

[3] ATLANTICWAVE-SDX. https://itnews.fiu.edu/wp-content/uploads/sites/8/2015/04/AtlanticWaveSDX-Press-Release_FinalDraft.pdf. (Cited on page 12.)

[4] BAILEY, J., PEMBERTON, D., LINTON, A., PELSSER, C., AND BUSH, R. Enforcing rpki-based routing policy on the data plane at an internet exchange. HotSDN, ACM. (Cited on pages 1 and 12.)

[5] BOTEANU, V., BAGHERI, H., AND PELS, M. Minimizing arp traffic in the ams-ix switching platform using openflow. (Cited on page 7.)

[6] CANINI, M., KUZNETSOV, P., LEVIN, D., AND SCHMID, S. A Distributed and Robust SDN Control Plane for Transactional Network Updates. In *INFOCOM* (2015). (Cited on page 12.)

[7] CASADO, M., KOPONEN, T., SHENKER, S., AND TOOTOONCHIAN, A. Fabric: A retrospective on evolving sdn. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks* (New York, NY, USA, 2012), HotSDN '12, ACM, ACM, pp. 85–90. (Cited on page 12.)

[8] COURSERA SDN COURSE, 2015. https://www.coursera.org/course/sdn1. (Cited on page 9.)

[9] DAVIE, B. S., AND REKHTER, Y. *MPLS: technology and applications*. San Francisco, 2000. (Cited on page 12.)

[10] DIXIT, A., HAO, F., MUKHERJEE, S., LAKSHMAN, T., AND KOMPELLA, R. Towards an elastic distributed sdn controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (New York, NY, USA, 2013), HotSDN '13, ACM, ACM, pp. 7–12. (Cited on page 12.)

[11] EXABGP. https://github.com/Exa-Networks/exabgp. (Cited on page 9.)

[12] FARINACCI, D., FULLER, V., MEYER, D., AND LEWIS, D. The locator/id separation protocol (lisp). Internet Requests for Comments, January 2013. http://www.rfc-editor.org/rfc/rfc6830.txt. (Cited on page 12.)

[13] FEAMSTER, N., REXFORD, J., SHENKER, S., CLARK, R., HUTCHINS, R., LEVIN, D., AND BAILEY, J. Sdx: A software defined internet exchange. *Open Networking Summit* (2013). (Cited on page 1.)

[14] GUPTA, A., VANBEVER, L., SHAHBAZ, M., DONOVAN, S. P., SCHLINKER, B., FEAMSTER, N., REXFORD, J., SHENKER, S., CLARK, R., AND KATZ-BASSETT, E. SDX: A Software Defined Internet Exchange. In *ACM SIGCOMM* (Chicago, IL, 2014), ACM, pp. 579–580. (Cited on pages 1, 2, 3, 4, 6, 7, 9, 10 and 12.)

[15] HASSAS YEGANEH, S., AND GANJALI, Y. Kandoo: A framework for efficient and scalable offloading of control applications. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks* (New York, NY, USA, 2012), HotSDN '12, ACM, ACM, pp. 19–24. (Cited on page 12.)

[16] ISDX GIHUB REPO. https://github.com/sdn-ixp/iSDX. (Cited on pages 2, 9 and 12.)

[17] ISDX HW TEST INSTRUCTIONS. https://github.com/sdn-ixp/iSDX/tree/master/examples/test-ms/ofdpa. (Cited on page 9.)

[18] KATTA, N., ALIPOURFARD, O., REXFORD, J., AND WALKER, D. Infinite cacheflow in software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (New York, NY, USA, 2014), HotSDN '14, ACM, ACM, pp. 175–180. (Cited on page 12.)

[19] KATTA, N., ZHANG, H., FREEDMAN, M., AND REXFORD, J. Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research* (New York, NY, USA, 2015), SOSR '15, ACM, pp. 4:1–4:12. (Cited on page 12.)

[20] KOPONEN, T., CASADO, M., GUDE, N., STRIBLING, J., POUTIEVSKI, L., ZHU, M., RAMANATHAN, R., IWATA, Y., INOUE, H., HAMA, T., AND SHENKER, S. Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 1–6. (Cited on page 12.)

[21] KOTRONIS, V., DIMITROPOULOS, X., KLÖTI, R., AGER, B., GEORGOPOULOS, P., AND SCHMID, S. Control exchange points: Providing qos-enabled end-to-end services via sdn-based inter-domain routing orchestration. (Cited on page 12.)

[22] LIGHTREADING. Pica8 Powers French TOUIX SDN-Driven Internet Exchange, June 2015. http://ubm.io/1Vc0SLE. (Cited on pages 1 and 12.)

[23] MAMBRETTI, J. Software-defined network exchanges (SDXs) and software-defined infrastructure (SDI), June 2014. Presentation at the Workshop on Prototyping and Deploying Experimental Software Defined Exchanges (SDXs). (Cited on page 1.)

[24] MONGODB. https://www.mongodb.org/. (Cited on page 9.)

[25] MONSANTO, C., REICH, J., FOSTER, N., REXFORD, J., AND WALKER, D. Composing software-defined networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2013), nsdi'13, USENIX Association, pp. 1–14. (Cited on page 6.)

[26] NoviSwitch 1132. http://noviflow.com/wp-content/uploads/2014/09/NoviSwitch-1132-Datasheet.pdf. (Cited on pages 4, 5 and 7.)

[27] Openflow 1.3 specifications. http://bit.ly/1eyrkxY. (Cited on page 7.)

[28] QuantaMesh BMS T3048-LY2. http://www.qct.io/Product/Networking/Bare-Metal-Switch/QuantaMesh-BMS-T3048-LY2-p55c77c75c159. (Cited on pages 7 and 9.)

[29] Richter, P., Smaragdakis, G., Feldmann, A., Chatzis, N., Boettger, J., and Willinger, W. Peering at peerings: On the role of ixp route servers. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (New York, NY, USA, 2014), IMC '14, ACM, pp. 31–44. (Cited on page 9.)

[30] RIPE. Ris raw data, 2015. https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data. (Cited on page 10.)

[31] Ryu SDN Framework. http://osrg.github.io/ryu/. (Cited on pages 1 and 9.)

[32] Sarrar, N., Uhlig, S., Feldmann, A., Sherwood, R., and Huang, X. Leveraging zipf's law for traffic offloading. *ACM SIGCOMM Computer Communication Review 42*, 1 (January 2012), 16–22. (Cited on page 12.)

[33] Smolka, S., Eliopoulos, S., Foster, N., and Guha, A. A Fast Compiler for NetKAT. In *ICFP* (2015). (Cited on page 6.)

[34] SQLite. https://www.sqlite.org/. (Cited on page 9.)

[35] Stringer, J., Pemberton, D., Fu, Q., Lorier, C., Nelson, R., Bailey, J., Correa, C., and Esteve Rothenberg, C. Cardigan: Sdn distributed routing fabric going live at an internet exchange. In *Computers and Communication (ISCC), 2014 IEEE Symposium on* (June 2014), IEEE, pp. 1–7. (Cited on pages 1, 2 and 12.)

[36] Tam, A.-W., Xi, K., and Chao, H. Use of devolved controllers in data center networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on* (April 2011), IEEE, pp. 596–601. (Cited on page 12.)

[37] Tootoonchian, A., and Ganjali, Y. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking* (Berkeley, CA, USA, 2010), INM/WREN'10, USENIX Association, USENIX Association, pp. 3–3. (Cited on page 12.)

[38] Whyte, S. Project CARDIGAN An SDN Controlled Exchange Fabric. https://www.nanog.org/meetings/nanog57/presentations/Wednesday/wed.lightning3.whyte.sdn.controlled.exchange.fabric.pdf, 2012. (Cited on page 4.)

[39] Workshop on Prototyping and Deploying Experimental Software Defined Exchanges. https://www.nitrd.gov/nitrdgroups/images/4/4d/SDX_Workshop_Proceedings.pdf. (Cited on page 12.)

[40] Yu, M., Rexford, J., Freedman, M. J., and Wang, J. Scalable flow-based networking with difane. *SIGCOMM Computer Communication Review 40*, 4 (August 2010), 351–362. (Cited on page 12.)