SDX: A Software Defined Internet Exchange

Arpit Gupta[†‡], Muhammad Shahbaz[†‡], Laurent Vanbever[⋆‡],
Hyojoon Kim[†], Russ Clark[†], Nick Feamster[†], Jennifer Rexford[⋆], Scott Shenker[◇]
[†]Georgia Institute of Technology [⋆]Princeton [◇]UC Berkeley
[‡]These authors contributed equally to this work

ABSTRACT

Deploying software-defined networking (SDN) at Internet Exchange Points (IXPs) offers new hope for solving long-standing problems in interdomain routing. SDN allows direct expression of more flexible policies, and IXPs are central rendezvous points that are in the midst of a rebirth, making them a natural place to start. We present the design of an SDN exchange point (SDX) that enables much more expressive policies than conventional hop-by-hop, destination-based forwarding. ISPs can apply many diverse actions on packets based on multiple header fields, and distant networks can exercise "remote control" over packet handling. This flexibility enables applications such as inbound traffic engineering, redirection of traffic to middleboxes, wide-area server load balancing, and blocking of unwanted traffic. Supporting these applications requires effective ways to combine the policies of multiple ISPs. Our SDX controller provides each ISP the abstraction of its own virtual switch and sequentially composes the policies of different ISPs into a single set of rules in the physical switches. Preliminary experiments on our operational SDX demonstrate the potential for changing interdomain routing from the inside out.

1. Introduction

The Internet's routing system is notoriously unreliable and difficult to manage. Network operators rely on arcane mechanisms to perform traffic engineering, prevent attacks, and realize peering agreements. These problems are deeply rooted in BGP's mode of operation, which has three particularly troubling limitations:

- **Routing only on destination IP prefix:** BGP selects and exports routes for destination prefixes, rather than customizing the routing decisions by application or sender.
- **Influence only over neighbors:** A network selects among BGP routes learned from neighbors, and exports selected routes to neighbors, rather than affecting end-to-end paths.
- **Indirect expression of policy:** Networks rely on indirect mechanisms like local-pref or AS prepending to influence path selection, rather than directing traffic to specific paths.

Software defined networking (SDN) could free interdomain routing from these constraints. First, SDN switches can forward packets based on many different packet-header fields, enabling flexible forwarding policies that go far beyond routing on destination prefix. Second, an SDN controller can receive control messages from remote networks on a bilateral basis (without the need for global standards), enabling the ultimate recipient of the traffic to influence the selection of the path. Third, an SDN controller can exert direct control by installing packet-processing rules in the data plane. Indeed, SDN offers new hope for fine-grained, flexible, and direct expression of interdomain policies.

However, simply having SDN-capable switches does not fix interdomain routing. Incremental deployment of alternative designs is a perennial problem in a global Internet with 50,000 independently operated networks and a huge installed base of BGP-speaking routers. Internet exchange Points (IXPs) are natural places to deploy new interdomain routing solutions. Changing even a single IXP can yield benefits for the tens or hundreds of ISPs meeting there. In addition, IXPs seek to innovate and differentiate themselves from competitors. The Internet has more than 300 IXPs worldwide—with more than 80 in North America alone—and some IXPs carry as much traffic as the tier-1 ISPs [2, 6]. For example, the Open IX effort seeks to develop new North American IXPs with open peering and governance, similar to the models taking root in Europe. The rise of video traffic, and growing tensions between content providers and access networks, puts IXPs on the front line of today's peering disputes. In short, not only are IXPs the right place to begin the interdomain routing revolution, but the organizations running these IXPs have strong incentives to innovate.

Kicking off an incremental deployment is impossible without compelling applications that can run at a single SDN-enabled IXP (SDX), without requiring changes to BGP. The next section presents several example applications, based on discussions with network operators and our own frustrations with BGP. These examples include security functionality (*e.g.*, dropping traffic not associated with an advertised route), remote-control peering (*i.e.*, allowing a content provider to control routing closer to the client), application-specific peering, redirection of selected traffic to middleboxes, and wide-area server load balancing. Later in the paper, we present prototypes of several of these applications.

An SDX also requires a control framework that can run these applications, and work with legacy BGP sessions, on behalf of hundreds of member networks. To minimize the rules in the switches, our SDX supports default forwarding of packets between member routers based on the BGP routing decisions, as in today's IXPs. For more sophisticated data-plane policies, our SDX controller applies *sequential composition* [15] to combine policies from multiple stakeholders into a single policy for the SDX switches. The SDX controller composes policies in an order that respects the business relationships between pairs of member networks, and any relationships the IXP has with remote networks.

Our plans to build an SDX have moved beyond paper design [1]: we have partnered with a large regional IXP that hosts many large providers (*e.g.*, Akamai) to deploy OpenFlow switches and our SDX controller. Our prototype system is already running at the IXP. The deployment includes two OpenFlow switches and a server that hosts several virtual machines—one for the SDX controller, and several others representing virtual peers. The deployment allows us to evaluate the new applications in a realistic setting in a real IXP, and to transition to switching traffic for real peers as our infrastructure matures and evolves. We expect our control architecture to evolve as we seek peers at the IXP and learn about what applications they want to deploy.

2. Limitations of Today's IXPs

In this section, we discuss how IXPs work today, and the resulting limitations on the policies they can support.

2.1 Traditional IXP Architecture

An IXP is a physical location where multiple networks meet to exchange traffic. An IXP is a layer-two network that, in the simplest case, consists of a single switch. Each member network physically connects one or more edge routers to the IXP, where each router port has a unique MAC address as well as a dedicated IP address from the IXP's own address block. Since all router ports belong to the same IP subnet, one member can direct traffic to another simply by sending a packet with the appropriate destination MAC address.

Two members can peer by establishing a Border Gateway Protocol (BGP) session between their respective edge routers, and applying local policies for selecting and exporting routes. Each BGP route has various attributes, including the IP prefix, the Autonomous System (AS) path, and the next-hop IP address of the neighboring router. Upon choosing a route, the member's router creates a forwarding-table entry that maps the destination IP prefix to (1) its output port connected to the IXP and (2) the destination MAC address of the chosen next-hop (resolved from the next-hop IP address using ARP). As such, the IXP switch does *not* need any information about IP prefixes, and simply forwards the packet based on the destination MAC address.

Rather than having a BGP session between each pair of members, IXPs often host a Route Server (RS) that acts as a sort of BGP multiplexer [8, 12]. Each member establishes one BGP session to the RS, and the RS applies all of the selection and export policies on each member's behalf, based on policies provided by the members. The RS sends each member one "best" BGP route (if any) for each destination IP prefix, subject to the export policy of the member that announced the route. The next-hop attribute corresponds to the IP address of this member's router port, rather than the RS itself, so the members can exchange data traffic directly. That is, the RS is purely a control-plane entity, with no participation in packet forwarding. In practice, IXP members use the RS for most BGP peering relationships, but may have dedicated BGP sessions for their most important peers [3].

2.2 What Today's IXPs Cannot Support

On the surface, an IXP already resembles an SDN, with the layer-two switches as the data plane and the route server as the controller. Indeed, route servers are natural places to deploy extensions to BGP policies and decision logic, though backwards compatibility with BGP is necessary to interoperate with member routers. However, there are significant constraints on the traditional IXP architecture. In particular, a route server does *not* determine the forwarding state in the switches, and conventional layer-2 switches do *not* act on header fields like the source IP address, TCP/UDP port numbers, and ToS bits. As a result, today's IXPs cannot support a wide range of data-plane policies, including:

Application-specific peering: ISPs are increasingly interested in "application-specific peering", where two members exchange traffic only for certain applications (e.g., video), identified by source IP address and/or port number. The IXP could implement this policy if its switches could forward a member's incoming packets to different peers for the same destination prefix, depending on the application.

Redirection to middleboxes: An IXP member could direct certain incoming traffic to a middlebox (e.g., sending port-80 traffic to a Web cache), by having the IXP switches encapsulate the packets or rewrite the destination IP address with the address of the chosen middlebox.

Traffic offloading: Suppose member A has a peering relationship with member B but not member C. If A selects a BGP route with a path that traverses B followed by C, the traffic would flow through the IXP (and B's link to the IXP) *twice* in traveling from A to B and ultimately to C. Forwarding the traffic directly from A to C would be more efficient, though ideally B could still charge A for the traffic that impinges on its relationship with C. Today's IXPs have limited control-plane mechanisms (e.g., "third-party next-hop") for traffic offloading, but not for monitoring the offloaded traffic.

Preventing free-riding: Today, an IXP member can easily "free ride" by directing traffic to another member that did not export a corresponding BGP route (e.g., a peer dumping traffic destined to a member's other peer). An IXP could easily block such traffic by installing a "drop" rule based on the input port, destination MAC address, and destination IP prefix, if the switches could support such rules.

Inbound traffic engineering: An IXP member may want to divide incoming traffic over multiple router ports, based on the sender, the peer, or the application. Destination networks that do not even connect to the IXP could perform in-

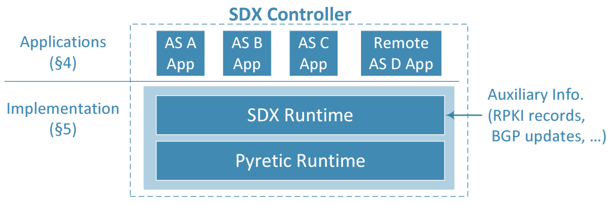

Figure 1: *The SDX controller architecture..*

bound traffic engineering if the switches could encapsulate each packet with the address of one of its homing locations.

Upstream blocking of DoS traffic: A destination network could have the IXP block or rate-limit suspected denial-of-service attack traffic close to the senders, if the switches can drop or rate-limit traffic based on the input port or source IP address, as well as the destination IP address.

Wide-area server load balancing: An IXP could implement a destination network's server load-balancing policy by splitting traffic over multiple data centers hosting a service. The IXP could announce an IP prefix on the network's behalf, and install rules that rewrite the destination IP address to direct different clients to different server replicas.

Running these applications requires IXPs to have more flexible switches, as well as effective ways to combine and enforce the traffic-handling policies of the stake-holders.

3. SDX Architecture

In this section, we describe the SDX architecture. Although we envision a SDX architecture involving multiple SDX controllers that coordinate with one another across multiple IXPs, we focus in this paper on the architecture of the SDX at a single IXP.

3.1 Overview

The SDX controller comprises a set of participant applications running on a runtime system built on top of Pyretic, as shown in Figure 1. The SDX runtime abstracts the details of the IXP (*e.g.*, virtual and physical topology, single or distributed switching fabric) from the participant applications. The SDX runtime also composes the different participants applications using sequential (">>") and parallel ("+") composition operators; during this stage, it also syntactically modifies participants' policies to ensure that participants' policies do not interfere or conflict with one another. Additionally, the SDX controller can incorporate auxiliary information such as resource public key infrastructure (RPKI) records, route server information, and so forth to affect routing decisions. We describe the details of these steps in Section 5.

3.2 Virtual SDX Abstraction

The SDX controller enables participant ASes to specify traffic handling policies as if they were the only one interacting with the system. To simplify policy composition, the

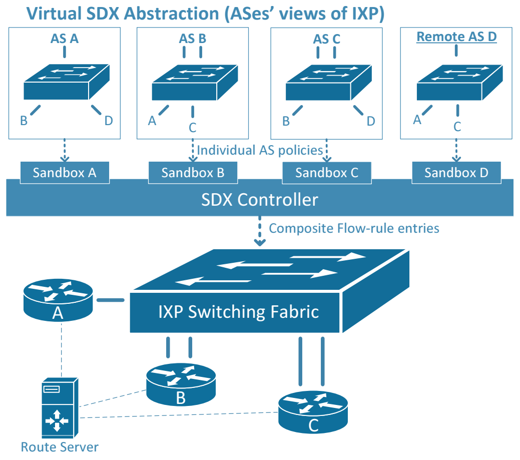

Figure 2: *Virtual SDX abstraction.*

SDX runtime presents an abstract topology to each participant; each participant sees the abstraction of managing a single switch, where other participants connect using a single link. This abstraction hides many details such as the underlying exchange topology which may involve distributed switches, the fact that other participants connect on multiple ports, on different switches or even in different physical locations. Moreover, this abstraction allows each AS to specify its own policies at the exchange independently from other ASes. In particular, since each AS is controlling his own switch, it does not be concerned about how or whether its policies will conflict with the policies of other ASes.

Each AS sees a different logical view of the IXP topology based on the other ASes it is allowed to communicate with (as is often dictated by business relationships). For example, in Figure 2, AS A sees a single virtual switch with connections to AS B and AS D; it does not see a link to AS C because it has no direct peering relationship with C. This *virtual SDX abstraction* prevents ASes from seeing parts of the network that they do not have permission to see or otherwise control. By default, the SDX controller forwards incoming traffic according to the AS's logical layer 2 topology defined by the virtual SDX abstraction and the routes from the BGP route selection process, which the SDX controller can compute based on input from the route server at the IXP. This default behavior has the lowest priority and is subsumed by any rules that participating ASes install. ASes can override this default behavior, but only with respect to their own virtual SDX topology. Because each AS sees only a restricted view of the IXP topology through its own virtual SDX, they cannot direct or control traffic for parts of the topology over which they have no authority.

Figure 2 shows the instantiation of an example SDX configuration where AS A and C do not have a direct peering relationship. AS A sees a virtual SDX abstraction that does not include AS C, and vice versa. The virtual SDX abstractions guarantee that A cannot direct or control any traffic received at C, nor can it send traffic directly to C. Similarly, A can only direct traffic to B's network by specifying the

```
AS A:  (match(dstip=ipB) >> fwd(B)) +
       (match(dstip=ipC) >> fwd(B)) +
       (match(dstip=ipA) >> fwd(outA1))

AS B:  (match(dstip=ipA) >> fwd(A)) +
       (match(dstip=ipC) >> fwd(C)) +
       (match(dstip=ipB) >> fwd(outB1))

AS C:  (match(dstip=ipA) >> fwd(B)) +
       (match(dstip=ipB) >> fwd(B))
       (match(dstip=ipC) >> fwd(outC1)) +
```

Figure 3: *Examples of SDX policies.*

Function	Section	How SDX Uses It	How Implemented
Redirection	4.1	middlebox redirection, inbound TE, traffic offloading	virtual SDX
Auxiliary info	4.2	application-specific peering, preventing free-riding	subsumption
Remote control	4.3	wide-area load balance, DoS squelching, path avoidance	virtual SDX

Table 1: *Example SDX functions, how they support different applications, and how they are implemented.*

virtual egress connection to B, as it does not know that B is actually connected at two physical points.

Because participant ASes see the other participants at the SDX through a virtual SDX abstraction, an AS doesn't even need to be physically present at the exchange to control traffic there. An AS could technically also control traffic flow even if it were not the sender or receiver of the traffic. We call this primitive *remote control*. For example, a content provider might want to use remote control to express a preference over the path that a video stream takes all the way to the client access network. To enable this function, the SDX controller can provide virtual SDX abstractions to participating remote ASes who have permission to alter traffic flows to and from certain ASes at the exchange. For example, in Figure 2, AS D is not physically present at the exchange, but it nonetheless sees a virtual topology through its virtual SDX that allows it to potentially control traffic concerning ASes A and C.

4. SDX Applications

We now describe three applications that the SDX enables. In Section 4.1, we describe how the virtual switch abstraction facilitates easy redirection of traffic through middleboxes. In Section 4.2, we show how the SDX controller can incorporate auxiliary information, such as routing information from route servers and other external sources. We also explain how the SDX controller allows operators to specify policies on subsets of traffic flows, subsuming the default forwarding behavior. In Section 4.3, we explain how the distributed virtual switch abstraction enables a remote AS to control certain forwarding behavior at an exchange. Although we focus on three applications in detail, we have identified multiple applications in each category of applications that we describe; Table 1 catalogs these applications.

4.1 Redirection: Middlebox Traffic Steering

The virtual SDX abstraction that the SDX controller exposes to each AS automatically enforces access control: An AS can send traffic to any device or endpoint that it can see in its respective virtual topology. Access control is thus implicit—an AS is free to send traffic to anything that is visible in its topology. This abstraction makes it easy to add network services and restrict their use to a particular AS or set of ASes: to add a service such as a middlebox, for example, an operator need only assign a virtual MAC address to the device that is reachable on that AS's virtual topology. To specify that a packet should be redirected through a middlebox, the receiver directs traffic to the virtual MAC address associated with the middlebox. The SDX controller maps the MAC address to the appropriate out physical port facing the middlebox. For example, suppose that AS A wants to redirect inbound port 80 traffic to the middlebox with virtual MAC address *M*. To do so, AS A sends the following policy to the SDX controller:

```
match(dstmac=macA, dstport=80) >> fwd(M)
```

The virtual SDX abstraction allows an AS to direct traffic to the middlebox, regardless of where the middlebox is located. The middlebox might be located in the participant's own network, in another AS's network [18], or at the IXP itself. If the middlebox's location changes, policies can remain the same as long as the middlebox's virtual MAC address remains the same. A middlebox could even use different virtual MAC addresses to indicate different packet processing pipelines on a single middlebox [4], thus allowing the AS to specify custom packet processing simply by changing the destination MAC address.

4.2 Auxiliary Information: Application-Specific Peering

As shown in Figure 1, SDX allows an AS to specify custom policies that subsume default forwarding behavior based on information from auxiliary sources. Route servers provide information such as default routes to each destination for each AS, thus allowing participant ASes to forward traffic according to the default BGP route selection process. Information from sources such as the Resource Public Key Infrastructure (RPKI) [10] can allow a participant to express policies that incorporate this additional information.

Participants can also write policies that subsume the default behavior. For example, in the case of application-specific peering, SDX might forward a participant AS's traffic to different peers depending on the application, even for the same destination IP address. In the case of application-specific peering, SDX forwards an AS's incoming packets to different peers for different applications, even it has a same destination prefix. Suppose AS A wants to send traffic to a specific peer, AS P, for Netflix's video traffic and all other traffic on the default BGP route. AS A can implement this feature with a policy that overrides forwarding behavior for the flow space corresponding to Netflix's video traffic:

```
match(srcip=177.72.244.0/21, srcport=80)
    >> fwd(P)
```

The SDX controller can compose this policy with the default forwarding policy for AS A's virtual SDX, based on AS P's BGP-advertised routes at the route server. AS A advertises its default routes via BGP to other ASes but uses the policy that it sends to the SDX controller to override the next-hop address for video traffic from Netflix, thus subsuming the behavior from the advertised BGP route.

The AS that receives the traffic, P, can explicitly write drop rules for any traffic from A that does not match this rule, but a more robust approach might be to create a virtual SDX abstraction using flow space, so that A cannot even see P for other traffic that does not have the specified source IP address and port.

4.3 Remote Control: WAN Load Balancing

The SDX controller enables *remote control*, whereby an AS can control exchange traffic even if it is not physically present at the exchange. The SDX controller also allows remote ASes to communicate their route preferences to multiple SDX deployments, thus enabling Internet-wide applications such as wide-area load balancing. For example, content providers can balance load across replicated services running on servers at multiple locations.

To implement wide-area load balancing, a content provider can advertise a single IP prefix corresponding for a service and send a policy to the SDX controller that directs a client's requests to the appropriate replica based on the client's IP address. The content provider can apply existing algorithms to divide the traffic based on client IP prefixes and ensure connection affinity across changes in the load-balancing policy [21]. For example, Google might advertise a single IP address, 74.125.1.1, for some service and direct specific customer prefixes to specific replicas (*e.g.*, sending clients on the west coast and east coast of the US to appropriate replicas):

```
match(dstip=74.125.1.1) >>
    (match(srcip=96.25.160.0/24) >>
        mod(dstip= 74.125.224.161)) +
    (match(srcip=128.125.163.0/24) >>
        mod(dstip=74.125.137.139))
```

This direct manipulation of forwarding tables allows a content provider to directly and quickly control redirection of traffic to replicas, in contrast to existing mechanisms, such as DNS-based load balancing. Before installing such a policy, the SDX controller must confirm that an AS has authority over the destination IP prefixes to which it is manipulating the destination IP address. The SDX can use auxiliary information (*e.g.*, from routing registries or the RPKI) to determine proper authority.

5. SDX Runtime Implementation

In this section, we describe the SDX runtime, which realizes each AS's virtual SDX abstraction by combining the policies of all participants into a single policy. Specifically, to realize policies such as those illustrated in Figure 3, the SDX runtime must generate a single policy that forwards packets from AS A with destination address `ipC` to AS C's physical output port. In addition to controlling forwarding behavior across the exchange, the runtime enforces isolation to ensure that participants cannot influence traffic or switch ports that do not belong to them. To realize these functions, the runtime performs two operations: Deriving a state machine to produce a virtual topology abstraction (Section 5.1), and composing the ASes' policies using sequential and parallel composition (Section 5.2).

5.1 State Machine for Virtual Topology

To create the virtual SDX abstraction for each participant, the SDX instantiates the virtual switch abstraction by defining the virtual ports that each participant can see. In the current implementation, this specification is manual and might be done by the SDX operator; we envision that the SDX controller might ultimately produce this configuration automatically through a syntactic translation of the policies that each of the ASes submits to the SDX controller. The controller passes this virtual SDX configuration to the Pyretic runtime, which generates the OpenFlow rules to realize the topology on the physical switches in the exchange.

Based on the virtual SDX topology specification, the SDX controller performs symbolic execution on packets using a finite state machine to determine how traffic should flow through the exchange to support the virtual SDX abstraction. The SDX controller associates a state with each packet at each step of forwarding. There are two types of packet state: (1) *switch states*, which indicate where in the virtual topology (*i.e.*, which switch) a packet is located; and (2) *output states*, which identify the output physical port onto which the packet should be forwarded. Symbolic execution continues until the FSM reaches a terminating out state. When a packet reaches a particular output state, the finite state machine terminates; the packet should ultimately be forwarded on the physical output port corresponding to the out state where the packet terminates.

Figure 4 illustrates the state machine that realizes the virtual SDX abstraction from Figure 3. A packet arriving at a physical input port is mapped to corresponding virtual switch state. Then, based on the destination IP address of the packet, the state machine transitions through a set of switch states until reaching an output state, which maps to the appropriate physical egress port. For example, a packet in state `A` with destination IP address `ipA` would transition to output state `outA1`, which would map to AS A's physical output port 1. The state machine also enforces business relationships: for example, there is no state transition from `A` to `C`, since ASes A and C do not have a business relationship. Note that the virtual SDX abstraction could also enforce relationships based on more specific parts of flowspace; in Figure 4, other parts of flow space are implicit wildcards, but transitions could also be allowed (or prevented) based on other parts of flow space. This mechanism could

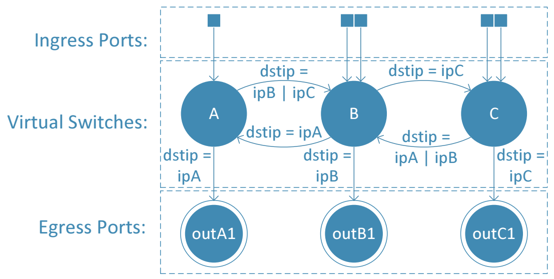

Figure 4: *Traversing virtual switches using a state machine.*

$$\texttt{ingress >>} \underbrace{\begin{pmatrix} P_1 \\ + \\ P_2 \\ + \\ \ldots \\ P_n \end{pmatrix} \texttt{>> ... >>} \begin{pmatrix} P_1 \\ + \\ P_2 \\ + \\ \ldots \\ P_n \end{pmatrix}}_{n \text{ times}} \texttt{>> egress}$$

Figure 5: *SDX sequential composition for n participants.*

be used to explicitly enforce application-specific peering arrangements.

5.2 Sequential Composition of Policies

The SDX runtime uses the result from symbolic execution on the state machine to sequentially compose the policies that realize the appropriate packet forwarding behavior through a sequence of virtual switches. Figure 5 shows this process. On ingress, the controller assigns the packet's `state` variable to a virtual switch (*e.g.*, in Figure 4, A, B, or C) based on the port on which the packet arrives. Similarly, the egress policy matches on the terminating output state variable (*e.g.*, `outA1`), and forwards the packet to the corresponding physical output port.

The intermediate steps compose the participants' policies based symbolic execution on the state machine. At each intermediate step, the controller applies the policies of all n participants at each step using parallel composition to ensure that one transition executes at each stage. The SDX controller extends each participant's policy with a match on its own state (*i.e.*, `match(state=A)` for A's policy) to ensure that policies are disjoint and only one participant AS's policy is applied at each intermediate step. The SDX controller converts `fwd` actions in each participant's policy into assignments to the appropriate `state` variable, extends the policy to forward packets based on the destination MAC address, and to implement default layer-2 forwarding. The composition of policies involves n intermediate steps. Typically, a packet should reach an output state in only a few steps. We currently limit n to the number of participants in the exchange to ensure termination; although it is currently possible that packets could loop through virtual switches in intermediate steps, limiting n ensures correct operation. Ultimately, additional loop detection techniques should be straightforward to incorporate.

6. Related Work

The most closely related work is Google's Cardigan project [22], which shares our broad goal of using SDN to enable innovation at IXPs. Cardigan runs a route server based on RouteFlow [17] and uses an OpenFlow switch to enforce security and routing policies. However, the project has *not* explored a general controller framework for composing a wide range of data-plane policies from multiple ASes, or remote control of forwarding from distant ASes.

Wide-area SDN research focuses *inside* a single AS, including intradomain traffic engineering [9, 11] and interdomain path selection [5, 17, 19]. These works do not explore how to combine policies from multiple ASes at an IXP. Other work proposes outsourcing end-to-end path selection to third parties [13, 14], but does not consider data-plane matching on multiple dimensions of header fields or ways to allow each AS to continue running BGP autonomously.

Our work builds on previous research on SDN programming languages [7, 15, 20], and particularly the topology abstraction and sequential composition features in Pyretic. However, these works do *not* propose abstractions for combining policies from multiple stakeholders at IXPs. In addition, our "state machine" approach to implementing the many-to-one virtual-switch abstraction is new and may be of independent interest.

7. Conclusion

SDX can break the logjam on long-standing problems in interdomain routing by enabling entirely new policies with fine-grained control over packet handling. The SDX supports policies that match and act on multiple header fields, and allow ASes to have remote control over the traffic.

Our SDX controller addresses many of the challenges of an SDN-enabled IXP. The virtual switch abstraction ensures that ASes cannot see or control aspects of interdomain routing outside of their purview. Sequential composition allows the SDX controller combine policies and resolve potential conflicts. Subsumption allows an AS to implement more sophisticated policies such as application-specific peering that override lower-level default forwarding behavior, enabling specialized control for certain subsets of traffic while ensuring correct default behavior for the remaining traffic.

Interest in the SDX is real: we have already deployed an initial SDX prototype in a large regional IXP, and we have released a preliminary version of the SDX controller [16]. Incremental deployment is possible because we assume ASes continue using BGP to exchange interdomain routes. Working within BGP does constrain our design, as SDX policies can occasionally cause the forwarding path to deviate from the AS path (as in application-specific peering). As demand grows for more flexible data-plane functionality, BGP should also evolve to support richer patterns (beyond destination prefix) and actions (beyond selecting a single next-hop). As SDX deployments begin to take hold, we may finally be able to fundamentally change interdomain routing and vector BGP on a path to the history books.

REFERENCES

[1] Two-page talk abstract, reference omitted to preserve anonymity.

[2] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, and W. Willinger. Anatomy of a large European IXP. In *Proc. ACM SIGCOMM*, 2012.

[3] AMS Internet Exchange. `https://www.ams-ix.net/ams-ix-route-servers/`, 2013.

[4] B. Anwer, M. Motiwala, M. bin Tariq, and N. Feamster. SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware. In *Proc. ACM SIGCOMM*, New Delhi, India, August 2010.

[5] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proc. USENIX NSDI*, 2005.

[6] Euro-IX Public Resources. `https://www.euro-ix.net/resources`.

[7] N. Foster, A. Guha, M. Reitblatt, A. Story, M. J. Freedman, N. P. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, et al. Languages for software-defined networks. *Communications Magazine, IEEE*, 51(2):128–134, 2013.

[8] R. Govindan, C. Alaettinoglu, K. Varadhan, and D. Estrin. Route servers for inter-domain routing. In *Computer Networks and ISDN Systems*, 1998.

[9] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *Proc. ACM SIGCOMM*, 2013.

[10] G. Huston and R. Bush. Securing BGP. *Internet Protocol Journal*, 14(2), June 2011.

[11] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hlzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. In *Proc. ACM SIGCOMM*, 2013.

[12] E. Jasinska, N. Hilliard, R. Raszuk, and N. Bakker. Internet exchange route server, February 2013. Internet Draft draft-ietf-idr-ix-bgp-route-server-02.

[13] V. Kotronis, X. Dimitropoulos, and B. Ager. Outsourcing the routing control logic: Better Internet routing based on SDN principles. In *Proc. HotNets Workshop*, pages 55–60, 2012.

[14] K. Lakshminarayanan, I. Stoica, and S. Shenker. Routing as a service. Technical Report UCB/CSD-04-1327, UC Berkeley, 2004.

[15] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing software defined networks. In *Proc. USENIX NSDI*, 2013.

[16] SDX Repository. `https://sites.google.com/site/sdxrepo/home/sdx-repository`.

[17] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proc. HotSDN Workshop*, pages 13–18. ACM, 2012.

[18] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: network processing as a cloud service. In *Proc. ACM SIGCOMM*, 2012.

[19] J. van der Merwe, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg, D. Knight, R. McMillan, D. Moloney, J. Mulligan, H. Nguyen, et al. Dynamic connectivity management with an intelligent route service control point. In *Proc. Workshop on Internet Network Management*, 2006.

[20] A. Voellmy, H. Kim, and N. Feamster. Procera: A language for high-level reactive network control. In *Proc. HotSDN Workshop*, 2012.

[21] R. Wang, D. Butnariu, and J. Rexford. OpenFlow-based server load balancing gone wild. In *Proc. HotICE Workshop*, March 2011.

[22] S. Whyte. Project CARDIGAN An SDN Controlled Exchange Fabric. `https://www.nanog.org/meetings/nanog57/presentations/Wednesday/wed.lightning3.whyte.sdn.controlled.exchange.fabric.pdf`, 2012.