SDX: A Software Defined Internet Exchange*

Arpit Gupta†, Laurent Vanbever⋆, Muhammad Shahbaz†
Sean P. Donovan†, Brandon Schlinker‡, Nick Feamster†, Jennifer Rexford⋆
Scott Shenker◇, Russ Clark†, Ethan Katz-Basset‡
†Georgia Tech ⋆Princeton University ◇UC Berkeley ‡USC

1. Motivation

Today's Internet routing is unreliable, inflexible, and difficult to manage. Most of the problems are rooted in the Border Gateway Protocol's (BGP) mode of operations, which has three particularly annoying limitations: It *(i)* routes only on destination IP prefixes; *(ii)* can influence (not control!) direct neighbors only; *(iii)* provides an indirect way of controlling forwarding paths.

We believe that Software Defined Networking (SDN) can easily remove each of these limitations by enabling: *(i)* flexible forwarding policies on many different packet-header fields; *(ii)* remote control as a SDN controller can receive control message from remote networks (on a bilateral basis); *(iii)* direct control on the forwarding paths by installing packet-processing rules directly in the data-plane.

Yet, simply deploying a few SDN switches here and there will not solve the problems of interdomain routing. Actually, improving Internet routing is–*and has always been*—a perennial problem in a world with 50,000 independently operated networks and a huge installed base of BGP routers.

We turn to Internet Exchange Points (IXPs) as a natural place to deploy new SDN-based inter-domain routing solutions. IXPs are topologically located at strategic positions—with most of the Internet traffic exchanged at around 300 IXPs around the globe. Even one deployment can therefore have a large impact. IXPs have also been particularly opened to innovations, a direct consequence of the fact that they are usually managed by a handful of people. Finally, IXPs are large switched networks and some of them have already started to roll out SDN-enabled equipments.

Our goal in this project is therefore to improve wide-area traffic delivery by designing, prototyping, and deploying Software Defined eXchanges (SDX) throughout the world. SDX platforms remove the pain out of BGP by enabling their participants to realize fine-grained inter-domain policies easily, *without requiring any change to the existing protocol*.

2. Challenges

Designing SDX requires addressing a lot of challenges. Some of them are:

Providing the right programming abstractions: In SDX, multiple competing participants with conflicting goals share the same data-plane. SDX needs the right abstractions to balance the desire for flexibility with isolation. Our SDX platform provides each participant with the illusion of owning a (virtual) SDN switch, where other participants connect using a single link. This abstraction hides many details such as the underlying exchange topology. Moreover, this abstraction allows each AS to specify its own policies at the exchange independently from other ASes.

Interacting with BGP to guarantee correctness: SDX cannot let its participants define policies in a vacuum, without regard to how they relate to the global routing system. The SDX therefore gives participating ASes ways to define forwarding policies *relative* to the current BGP routes and ensure safe interaction with the existing routing system.

Ensuring scalability: SDX needs to support hundreds of participants, hundreds of thousands of IP prefixes, and policies that match on multiple packet-header fields—all while using FIB constrained SDN switches. To enable scalable operation, the SDX platform minimizes both the *space* (# of flow rules) and the *time* (time for policy compilation) complexities. SDX leverages existing control-plane signaling mechanism to reduce the data-plane state and domain-specific knowledge to speed up the compilation of policies.

Enabling a realistic deployment: We have built a SDX prototype and created multiple example applications [3] to demonstrate that our prototype scales to many participants, policies and prefixes. In this demo, we'll present one such application highlighting various capabilities of our platform.

3. Implementation

The SDX consists of a switching fabric, BGP edge routers, and a SDX controller. Edge routers are unmodified and they simply establish normal eBGP sessions with the SDX controller, which also acts as a BGP Route Server. Participants send their policies written in Pyretic [2] to the SDX controller as JSON messages. The SDX controller implementation is divided into two processing pipelines (see Figure 1). One dealing with SDX policies (*Policy Compiler*), and the other with BGP routes (*Route Server*).

Policy Compiler takes Pyretic based policies of all the participants as input and produces aggregate forwarding rules. In the process, it augments the policies to *(i)* enable isolation; *(ii)* reduce data-plane state; *(iii)* and integrate with advertised BGP routes. Based on the virtual switch abstraction, the

*Also appears in the main program [1].

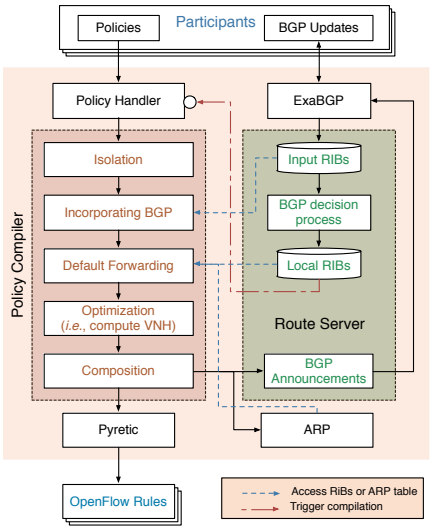

Figure 1: *The SDX controller implementation is based on two communicating pipelines: one dealing with SDX policies, and another one dealing with the processing of BGP routes.*

policy compiler isolates participants policies by augmenting them with additional `match` statements based on participant's physical/virtual ports. The compiler then restricts each participant's outbound policies according to advertised BGP routes. It also analyses each participant's policies to identify sets of prefixes sharing the exact same forwarding behavior and assigns one tag (Virtual Next-Hop) to each of them. It then rewrites the policies, replacing match on destination prefixes with match on tags instead. In the SDX, we use IP Next-Hop as tags and rely on BGP to provision (transparently) these tags in the participant edge routers directly. The SDX controller therefore also has an ARP-resolver to respond to queries for Virtual Next-Hop IP address. As a final step, the compiler composes policies of all the participants together to produce a set of forwarding rules. The policy handler also receives events from Route Server and from the participants whenever there is change in the best path for a prefix or in participant's policy, respectively.

Route Server performs the default functionality of a typical route server. It receives BGP advertisements from all the participants and computes the best paths on their behalf. It also: *(i)* provides the advertised route information to the compiler pipeline; *(ii)* advertises virtual next hops for each prefix rather than the physical ones; *(iii)* sends an event to the policy handler whenever best route to a prefix changes.

4. Demonstration Outline

The main goal of our demonstration is to highlight four major capabilities of SDX: *(i)* enabling participants to control the traffic flows based on the portions of flow space other than destination IP prefixes; *(ii)* enabling dynamic composition of participant's policies into a single set of coherent flow rules; *(iii)* minimizing the data-plane state by leveraging the existing BGP control-plane; *(iv)* guaranteeing correct forwarding behavior in sync with advertised BGP routes.

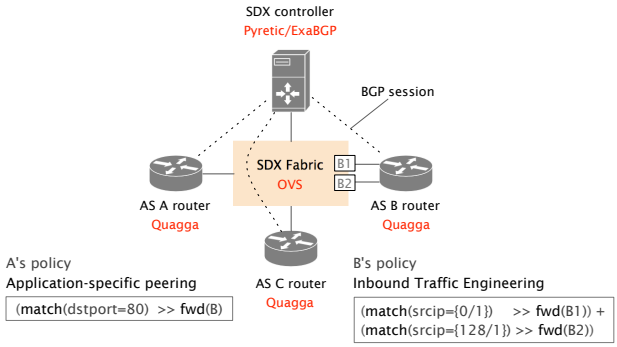

Figure 2: *Demonstration setup. By default, all traffic is sent to C. When A provides its policy, all HTTP traffic for the prefixes reachable via B is sent to B. This traffic is then load-balanced according to B's inbound policy.*

Figure 2 depicts our demonstration setup. It consists of a SDX controller (which includes a route server) connected to an Open vSwitch software (OVS) switch. We also have three BGP edge routers, running the Quagga routing engine, connected to the switch. These BGP routers belong to the participating ASes: AS_A, AS_B, and AS_C. AS_B's router is connected via two input ports at SDX. AS_B and AS_C advertise prefixes p_1-p_5 and p_1-p_{10} respectively to the SDX controller. For each prefix, the route server chooses the route via AS_C as best. Policies are configured as shown in Figure 2. AS_A sends uniformly distributed traffic for different sets of source/destination IP addresses and destination ports.

In our demonstration, we will start without any SDX policies. We will observe that all the traffic from AS_A exits via AS_C. Next, we will send AS_A's application specific policies to the controller. Now, we will observe AS_A's *HTTP* traffic leaving the switching fabric via $B1$. This demonstrates that SDX enables control over traffic flows beyond destination IP prefixes. We will also see that non-HTTP traffic for prefixes p_1-p_5 and all the traffic from prefixes p_6-p_{10} still exits via AS_C. This demonstrates that the SDX controller is in sync with the advertised BGP routes. We will now apply AS_B's inbound traffic engineering policies and will observe that half the *HTTP* traffic leave via port $B1$ and the remaining half via $B2$. This demonstrates how SDX composes policies of two participants with disparate high-level goals. In addition to this, we will also show the flow tables to demonstrate the efficacy of using Virtual Next Hops to reduce the number of flow rules in the data-plane.

REFERENCES

[1] A. Gupta, L. V. M. S. S. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett. Sdx: A software defined internet exchange. In *Proceedings of the 2014 ACM SIGCOMM Conference*, Chicago, Illinois, Aug. 2014. ACM.

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

[3] Software Defined Internet Exchange Point (SDX) Project. `https://github.com/sdn-ixp/`.