Services Load Balancer for a Software Defined IXP

Miguel Oliveira Rodrigues dos Santos

Abstract—The advent of Software Defined Networking (SDN) technology opens new opportunities to scale interdomain routing. Industrial Scale SDX (iSDX) is an implementation of an Internet Exchange Point (IXP) using SDN to emulate the usual features of an IXP. For a new technology to be useful, it cannot simply emulate the functionality of an old one; it has to excel at the old tasks and bring in new capabilities. Our work proposes an extension to iSDX, in order to provide added value to the customers, with an implementation of a Services Load Balancer.

Index Terms—Routing, Load Balancing, Ryu, Software Defined Networking (SDN), Internet Exchange Point (IXP), Industrial Scale SDX (iSDX)

1 INTRODUCTION

In the early days of the Internet, scaling was not as big of a problem as it is today. A notorious episode known as a flag day happened in the first day of 1983, when network operators made the switch from Network Control Program (NCP) to Transmission Control Protocol over IP (TCP/IP) only two years after the first Request For Comments (RFC) describing TCP/IP were released. The modern scale of the Internet makes global coordination between network operators rather impossible. An infamous case is the switch from Internet Protocol, version 4 (IPv4) to Internet Protocol, version 6 (IPv6): even though IPv6 has been around for over a decade, it is still not as ubiquitous as we would desire.

As with all innovations in the Internet, being evolutionary is necessary requirement. Revolutionary solutions are impractical to implement since backwards compatibility is imperative. Even if SDN was completely ready to take over the whole Internet, we would still have to go through an evolutionary process, carefully introducing SDN in the ecosystem while keeping backwards compatibility. This thesis aims to contribute towards a faster adoption of SDN technology in the interdomain world.

The Internet Exchange Point (IXP) ecosystem handles a big part of the traffic in the Internet, making it the ideal place to start an incremental evolution of interdomain routing. An SDN powered IXP can easily implement some features that a classic IXP cannot implement, since leaving routing to Border Gateway Protocol (BGP) leads to loss of flexibility. [3] analyses several advantages of an SDN approach to peering such as application-specific peering.

In [4, p. 4], the authors propose an implementation of a Software Defined IXP (SDX) and state that “Participants who are physically present at the IXP but do not want to implement SDX policies see the same layer-2 abstractions that they would at any other IXP”, which is exactly why IXPs are a good place to start SDN deployment in the Internet. Another solution is proposed by the same authors two years later in [5], which solves most of the scalability problems in [4].

In Section 2 we will analyze this iSDX proposal [5], looking at both the concept and the implementation. To understand the architecture of iSDX it is useful to start by introducing its predecessor, SDX [4].

In Section 3 we implement a load balancing service in the iSDX as a means to prove that
the interdomain routing ecosystem can benefit largely from SDN while still keeping current protocols unchanged.

2 RELATED WORK

We start by analyzing [4], an architecture proposed in 2014 which served as a stepping stone for [5]. As we will see, the initial architecture first described in [4] had severe scalability problems which were later addressed in [5].

2.1 SDX: A software defined IXP [4]

From each peer’s point of view, the IXP should be a single logical switch. Each peer sees a different logical switch to which it may write policies (forwarding rules for the logical switch). All policies are compiled into a set of forwarding rules that is written to the physical switches. The SDX includes a route server which collects the routes advertised by each participant. The best route to each prefix is announced to each participant. Participants can override this default route as long as the destination’s prefix was announced by the next-hop Autonomous Systems (AS). This default route behavior ensures interoperability with any system that would work with legacy networks.

Efficient compilations of all ASs’ policies is a major challenge, since the context requires time- and space-efficient computations. Compilation is achieved through the combination of several specially crafted techniques:

Policy transformation: syntactic transformations on the policies written to the SDX.

1) Restricting policies according to the virtual topology: Guarantees that only the allowed logical paths may carry traffic. Participant isolation is achieved by guaranteeing that it will only act upon policies in its own virtual switch.

2) Augmenting policies with BGP-learned information: Consistency with BGP-learned reachability information must be achieved, since SDX guarantees interoperability with the existing protocols. Participants are therefore only allowed BGP-learned routes. The SDX generates a filter that is applied to the participant’s outbound policy.

3) Extend policies to default to using the best BGP route: Only the traffic that matches a participant’s policy should be affected by it. All the remaining traffic should be forwarded according to the default routing decisions.

4) Composing the policies of all the participants into one: Upon entering a virtual switch the outbound policy of the switch is applied followed by the inbound policy of the following virtual switch. Combining participants’ policies is achieved through enforcing this logic after having the complete policies for each participant.

Reducing data-plane state: Augmenting policies based on BGP-learned information may lead to a huge number of forwarding rules. There are about half a million IPv4 prefixes in today’s global routing system, and IXPs tend to have hundreds of participants. Augmenting each participant’s policy with each prefix it receives is not feasible, since the maximum number of rules imposed by hardware limitations would not be enough. Data-plane scaling is achieved through grouping prefixes into equivalence classes, since a participant’s policy will usually treat a large number of prefixes in the same way. These classes are called Forward Equivalence Class (FEC) and represent a group of prefixes that are handled in the same manner. The FEC is written in the packets by the participant’s router, thus eliminating the need for the core system to store each participant’s full table. On the data-plane level, the FEC is simply a Virtual MAC Address (VMAC) address. The control plane also needs to have some mechanism to instruct the participant’s router on which groups of prefixes to tag with which VMAC. The way SDX handles this challenge is by pre-computing the FEC according to the participants’ policies, and populating an Address Resolution Protocol (ARP) table within its own ARP server so that it associates FECs with the desired next hops.

Reducing Control Plane computation: control plane computation is necessary, but scalability concerns require that the amount of computation should be minimized. The three different routines intrinsic to this architecture are:
1) Computing next hops
2) Augmenting participants’ policies
3) Compiling policies into forwarding rules

The authors of the paper decided to focus primarily on the second aspect, since it is the most resource-intensive routine. The article explains the optimizations in detail. A key flaw in the mechanism is that the SDX’s route server has to recompile the participant’s policies whenever a BGP update results in changes for a best route for a prefix. This is, of course, prohibitive. In a modern IXP, changes in the best routes for a prefix happen often, and it would not be practical to recompile participants’ policies every time this occurred.

2.2 iSDX: An Industrial-Scale Software Defined Internet Exchange Point [5]

iSDX was built with scalability in mind, thus minimizing policy compilation time and forwarding table sizes. [5] analyzed the performance using data from one of the world’s largest IXPs and found that this solution scales to the current IXP scene. Performance analysis measurements confirm a decrease by two orders of magnitude in both policy compilation time and forwarding table sizes, when compared to the implementation described in Section 2.1. Most importantly, new mechanisms were also introduced to eliminate the need to recompile all policies whenever a BGP update results in a change in a participant’s default next-hop. So not only is the policy compilation faster, it also occurs less frequently.


2.2.1 Scaling

Scaling the control plane began with the idea that each participant should compile its own policies, which allows for a more effective compression. This approach reduces computation in the controller and eliminates dependencies between each participant’s policies that existed in the previous solution after compiling all policies together. A participant that changes his inbound policy will no longer cause recompilation of all policies of participants who forward traffic to his network.

The approach to determining the FEC also evolved: instead of forcing the core to compute the set of equivalence classes, this responsibility is offloaded to the participants’ routers who compute the FECs for each participant separately.

2.2.2 Decomposing the IXP fabric into four tables

In the previous design, policy composition involved combining each inbound policy with an outbound policy, which lead to explosive growth in the number of entries in the forwarding tables caused by a cross-product of policies upon compilation. The cross-product can be avoided through the use of multiple tables, which OpenFlow v1.3+ already supports in the shape of multiple match-action stages.

First, two tables are created: one inbound table for the inbound policies, one outbound table for outbound policies. With this approach the cross-product problem is still not completely avoided but rather minimized. As an example: if n ASs have the same outbound policy regarding a property of the outgoing traffic, that information will take n entries in the outbound table. Policies are still cross-products between the match and action parts of the policy, as illustrated below.

In this illustration (Listing 1), we will suppose that participants 1...n have a certain outbound behavior (an action) that applies whenever certain condition regarding its header is fulfilled (a match).

For each of the n participants, an line must be written to the outbound table.

Listing 1. Sample outbound rules

1) match(src = 1) >> action(ACTION)
2) match(src = ...) >> action(ACTION)
3) match(src = n) >> action(ACTION)

Introducing two new tables (input table and output table) solves this cross-product problem. How this works is not obvious. Figuring this out required reading the source code since the original paper abbreviates the explanation to a
SERVICES LOAD BALANCER FOR A SDX

point where the purpose of the two new tables is not entirely clear.

- **Input table**: A packet leaving an AS towards the iSDX passes through the input table, where its source Media Access Control (MAC) address is rewritten based on the packet’s incoming port. The value of that field is then used to multiplex the packet when going through the outbound table (like an index in a hashtable).

- **Output table**: Rewrites the destination MAC address and indicates the port through which the packet should leave the iSDX.

In short, the output table undoes the changes that the input table did. These changes are useful for the matching stages of the outbound and inbound tables.

The modifications in the packet headers which occur in the input table allow the inbound and outbound tables to use the source MAC addresses to match entire groups of participants instead of individual sources. Notice that the MAC address will contain more information in a special encoding detailed in Section 2.2.3. The encoded information includes not only the FEC, but also the specific peer who sent the message, so that this value can be restored by the actions in the output table.

To illustrate the benefits of this grouping, the previous example is re-written using this approach (Listing 2).

Listing 2. Usage of a group tag

```
match(src = GROUP) >> action(ACTION)
```

Notice how the number of entries in the table was reduced drastically. The cross-product problem is therefore completely mitigated when this four-table architecture is implemented.

To aid the understanding of the system, consider the graphical representation in Figure 1 and the following description of a packet’s history through the iSDX. AS_A’s controller selects a route for a given AS and advertises it through BGP. AS_A’s router will issue an ARP query for the next-hop corresponding to the announced path. AS_A’s controller will reply with a virtual MAC address that will be set as the packet’s destination. Upon passing through the input table, the source MAC address is rewritten to indicate the packet came from AS_A (and of course, the FEC). If the fields source MAC and virtual next-hop match an index in the outbound table, the destination MAC is overwritten (that is, default BGP routes are overwritten by the outbound policy). Whether or not an outbound policy was applied, the packet is forwarded to the inbound table where the destination MAC may be overwritten if it matches one of the destination AS’s inbound policies. Finally, the output table will replace the destination MAC address by the destination machine’s actual physical address and the underlying forwarding mechanisms should work as expected.

Handling traffic based on arbitrary fields also helps decreasing ARP traffic within the iSDX. Since each participant only needs to be informed of the VMAC that corresponds to its own FECs, the iSDX can match the Ether-Type field of the Ethernet Frame in order to filter ARP traffic and make it unicast. Every time a participant’s controller decides on a new virtual next-hop, it issues a gratuitous ARP response that is forwarded only to the participant’s router. Participants may still make use of the usual ARP query, but the request is forwarded to the ARP relay instead of being broadcast to all participants’ routers.

2.2.3 Statically encoding BGP routing information in a tag

SDN policies need to respect BGP information in order to guarantee correctness. Although necessary, this raises problems regarding the large number of forwarding table entries in the participant’s outbound table and the need to recompile policies every time a BGP update results in a default next-hop change. To mitigate this problem, BGP information was completely separated from SDN policies. Reachability information is therefore not stored in the core of the iSDX (unlike the previous version of that project which, as previously mentioned, uses a route server). Packets are tagged in order to include the necessary BGP information. The tag identifies:

1) Default next-hop (hereinafter next-hop bits)
2) Set of ASs that advertise routes to the destination of the packet (hereinafter reachability bits)

**Usage of the tag:** The tag is learned by the participant’s router through ARP, as it is calculated by the participant’s controller. If no SDN policies are to be applied to the packet, the IXP forward traffic according to the default next-hop. In case of non-default behavior (i.e. when an SDN policy applies to the packet), the iSDX uses the reachability bits to statically check whether or not the next-hop corresponding to the next-hop bits has advertised the destination prefix.

Inside the IXP, a table stores a tuple (next-hop tag bits, next-hop AS) for each next-hop AS. The existence of this table makes it possible to enumerate participants in a way that can be encoded in the tag. If the default next-hop changes as part of BGP’s normal operation, only these table entries need to be changed in the controller (and consequently, the participant router’s next-hop bits change after a gratuitous ARP is sent by the participant’s controller). Policies in the tables remain unchanged. Thus, since there is no need to recompile the policies, static encoding of reachability information enhances the performance of the system.

**Syntax of the tag:** The tag will be written to the destination MAC address field in the Ethernet frame, so that leaves 48 bits for the implementation to encode the information. One of the bits is immediately left out since it is reserved for multicast, as per RFC 7042 [8]. Encoding is engineered as follows:

- **Next-hop:** Assuming a reasonable number of participants for an industrial IXP is \( N = 2^{10} \), then \( \log_2(N) = 10 \) bits are enough to enumerate the participants. This leaves 37 free bits to be used for the remaining information.
- **Reachability encoding:** The goal here is to minimize the number of bits used to identify a group of participants that announce a given next-hop prefix. Bear in mind that at this point, there are only 37 bits available to contain this information. The suggested approach finds a minimal set of bitmasks that represent groups of participants and then uses another bitmask to select which of the participants in the group announce the prefix.

The authors examined real-world data from a large IXP which revealed the most announced prefix was announced only by 27 participants (which corresponds to using 27 bits in the bitmask). The ID of each bitmask uses a number of bits that depends on the number of bitmasks. The same experimental data revealed 62 bitmasks were enough to contain all groups, thus encoding the bitmask ID uses \( \lceil \log_2(62) \rceil = 6 \) bits.

The MAC address field still has 4 free bits to be used in the future, in case any of
these assumptions change.

2.3 Architecture of the iSDX
A representation of the architecture of an iSDX was already illustrated in Figure 1.

The IXP controller is responsible for installing forwarding table entries in the underlying switching fabric. The rules which translate into forwarding table entries are static and therefore, computed only at initialization. Communication between the IXP controller/participant controllers and the physical switches is intermediated by the fabric manager, which abstracts the topology of the network and presents a virtual topology to the controllers.

The IXP runs a route server that establishes BGP sessions with the participants’ routers but instead of performing route selection like a common route server, it simply relays BGP information to the participants’ routers since that is where route selection happens in this architecture.

The participants’ controllers run an update handler and an ARP handler. The former determines whether or not the policies should be updated, and may update the corresponding participant router’s ARP table by sending out a gratuitous ARP. The latter receives ARP messages from the participants’ routers and determines the corresponding ARP reply.

By using data from a large IXP and assuming what typical SDN policies might look like, the authors measured performance of the final implementation. A thorough analysis is found in [5, p. 9].

In short, policy compilation time and forwarding table size were reduced by several orders of magnitude when compared to the previous implementation. Using real-world data, the authors conclude that iSDX can in fact scale to industrial level.

2.4 Critique of iSDX design
To make the iSDX scalable to industrial-IXP sizes, several compromises were made. In particular, some responsibility was given to the participants’ routers and controllers. In this section analyses some of these compromises and potential pitfalls that come along with them.

2.4.1 Security: is offloading responsibilities to participants’ routers a good idea?
Since the software running on participants’ routers and controllers is controlled by each participant, the systems should be prepared to handle specially crafted messages aimed at exploiting the system. In this section we explore two attacks to this architecture that involve modifications only in the parts of the architecture that a single malicious participant controls.

Sending traffic along unannounced paths: Since a participant controls the reachability information that goes in the MAC headers, this information can be manipulated. The tag in the destination MAC address of an Ethernet frame may be forged in order to include a participant that has actually not announced a path to the destination. If either:

- The next-hop field corresponds to the destination whose reachability information was forged, or
- Some SDN policy in the iSDX forwards traffic to that destination,

then the iSDX will forward the traffic without noticing the attack. According to the encoding scheme seen in Section 2.2.3, forging this reachability information is as easy as flipping a bit in the reachability part of the destination MAC address.

Bypassing other participants’ inbound policies: When an Ethernet frame travels through the iSDX, the validity of matching policies is checked using the reachability information in the Ethernet frame’s headers. If a policy forwards to a destination that is unreachable according to the encoded reachability information, then it is simply ignored. While this approach allows the iSDX to adapt to changes in routing, it also allows a participant to bypass another’s inbound policies.

Consider two ASs connect to the iSDX. Both routers belonging to AS$_B$ announce the prefix 140.0.0.0/24, but an inbound policy forces traffic directed to 140.0.0.100 to be forwarded to AS$_{B1}$. Under normal circumstances, traffic from AS$_A$ aimed at 140.0.0.100 would be forwarded to AS$_{B1}$ after passing by the iSDX’s fabric. But
if something causes the iSDX to ignore the policy that is responsible for this behavior, then the default next-hop is followed.

Let’s assume \( AS_A \) wants to force traffic to go through \( AS_{B2} \). With that goal in mind, \( AS_A \) forges the information in the reachability bits so that it does not list \( AS_{B1} \) as a participant who announces the prefix \( 140.0.0.0/24 \). The inbound policy will not take effect and the default next-hop will be respected. A simple malicious modification in the header may therefore result in \( AS_A \) successfully bypassing \( AS_B \)’s inbound policy.

It is clear that offloading responsibilities to participants’ routers creates new security challenges. iSDX brings features at the cost of new security challenges that need to be addressed.

2.4.2 Scalability: Limitations concerning the number of participants

Overwriting the destination MAC field is used to encode relevant routing information: the next-hop tag bits and the reachability bits identifying the participants who announced a path to the prefix that contains the non-default next-hop.

One possible problem with this approach is that it makes assumptions about the number of users of the iSDX: to determine how many bits are used to encode the number of participants, the authors assume 1024 participants is enough. While 1024 is a reasonable number of participants in current IXPs, this is still a limitation on the maximum number of participants in the iSDX.

The method followed to encode reachability information also presents similar limitations. Instead of assuming a maximum number of participants, the method assumes a maximum number of participants announcing a same prefix.

The paper suggests “Using a different (or custom) field in a packet header might also be possible if these numbers grow in the future.” [5, p. 8]. The use of a separate tag would have an impact in the overall complexity of the system, since the restraints caused by the limited number of bits in the MAC fields are a significant problem in this architecture.

2.4.3 Routing correctness

While breaking pure-BGP routing increases the flexibility of inter-domain routing, it may also bring correctness problems. The existence of two SDXs is enough to open the possibility of traffic being caught in a persistent forwarding loop.

To illustrate the problem, the article shows a simple example (see Figure 2) of a persistent forwarding loop in a system, caused by two seemingly unrelated policies.

![Figure 2](image.png)

Figure 2. A forwarding loop caused by two seemingly unrelated policies in two different SDXs [9, p. 1].

In Figure 2 there is a forwarding loop for Hyper Text Transfer Protocol over SLL (HTTPS) traffic destined to prefix \( p_1 \). Since this is Ethernet traffic, there is no Time To Live (TTL) field so traffic keeps on looping without any reasons to stop.

3 IMPLEMENTING A LOAD BALANCING SERVICE IN THE iSDX FRAMEWORK

There are two possible ways to implement load balancing in this framework: proactive policies (writing policies to the hardware beforehand) and reactive policies (sending packets to the controller, which then creates the rules that should be written to the controller). When a stream of one or more packets is sent through a load-balanced service implemented using a reactive approach, the matching phase results in a table-miss which causes the packet to be forwarded to the controller. The controller then inspects the packet and creates a new rule that is written to the hardware. The packet is sent towards its destination and the subsequent
packets in that stream will match the newly written forwarding table entries.

Since the controller may quickly become a bottleneck, the number one priority when developing a reactive implementation should be to reduce the load in the controller.

As discussed in Section 2.2.2, the iSDX uses four tables to match traffic and direct it towards its end.

On a first look, adding each participant’s load balancing rules to the inbound table sounds reasonable. This would be the simplest approach, though it would make it impossible to have load balancing policies and inbound policies at the same time.

The solution is to add a new table to the packet processing pipeline - the load balancing table. This table will only concern layer 3 information, as its purpose is to replace an anycast Internet Protocol (IP) address by a node’s IP address.

The order of the tables in the pipeline is extremely important, as seen in Figure 1. The key consideration is that it should be positioned such that it allows participants to write outbound policies that direct traffic to the load-balanced service’s anycast IP address. Thus, the table should be inserted immediately after the outbound table, so that a participant who owns load-balanced services can write load-balancing policies that match on the IP addresses of the machines running the service and not just on the anycast IP address.

With the implementation of these load-balancing capabilities, the iSDX uses a total of 5 tables.

3.1 Load distribution methods

When a client initiates session establishment, the iSDX should dynamically write a rule to the hardware that directs its traffic to one of the nodes running the desired service.

The distribution algorithm should choose a destination IP address in a fairly uniform manner. A weighted version of the algorithm may increase the relative frequency of a given outcome, which translates into increasing the usage of a certain end server.

Four different methods were tested and proven to achieve fairly even load distribution when picking IP addresses off of the list of aliases of the load-balanced service:

- **Round Robin**: The algorithm returns the first, second, third, ..., n-th item in the list, the wraps around the edges and starts from the beginning.
- **Weighted Round Robin**: Similar to the pure Round Robin, except an item is not returned just once in each round; each item is returned a number of times that is proportional to its weight.
- **Random**: A random IP address is used among those in the list.
- **Weighted Random**: Similar to the pure Random implementation, except the likelihood of an item being chosen is proportional to its weight.

Weights are useful, e.g., in scenarios in which some serving nodes are more powerful than others. A slightly simplified implementation of the Round Robin load balancer is exemplified in Listing 3:

```python
class RRLoadBalancer():
    def __init__(self, anycast_ip, alias_list):
        self.index = 0

    def get_ip(self):
        ip = self.alias_list[self.index]
        self.index = (self.index + 1) % len(self.alias_list)
        return ip
```

The remaining implementations are equally simple.

4 Validation and Analysis

A small bash script was written to simulate 28 Hypertext Transfer Protocol (HTTP) connections between a client and the HTTP server running in the anycast IP address. The number of sessions was chosen to match exactly 4 rounds through the 7 servers.

```bash
#!/bin/bash
for i in `seq 1 28`; do
    wget http://<anycast_ip>:8885/ &
done
```
The virtual IP address established for the service is 140.0.0.222 and behind it, seven machines run the HTTP service.

It is expected that the evolution of the number of sessions in each server varies according to the approach each method takes. Validation will be performed through the visual representation of the evolution of the number of sessions established with each server.

**Round Robin:** It is expected that the load is shared between each of the seven machines evenly with no variance. Each machine should log exactly one connection before the first machine receives the second connection.

Figure 3 was plotted using the data logged by the servers when the client runs the bash script.

As expected, the cumulative number of sessions is steady, increasing by one unit at each round.

**Weighted Round Robin:** The weights used in this example are [1, 2, 3, 4, 5, 6, 7].

It is expected that the first round of the algorithm results in one session being established with each server. The second round will establish sessions with all servers except for the first one. The $7^{th}$ round will consist of a single session with the last machine. After a period of seven rounds, the cycle repeats.

Figure 4 empirically demonstrates the expected behavior.

**Random:** In a truly random distribution, the number of sessions established with each server must be approximately the same. Figure 5 represents empirical data regarding random distribution of sessions through the servers.

While the results seem to match the expected behavior, it is impossible to be certain about the correctness of the implementation without more data points. An example with a higher number of sessions (in this case, 2000) did better job at evaluating the correctness of the algorithm. The fraction of sessions established with each server $\approx \frac{1}{7}$.

**Weighted Random:** The weighted random approach results in a number of sessions that is proportional to the weights attributed to
each server. As in the previous example, the weights used were \([1, 2, 3, 4, 5, 6, 7]\). Notice in Figure 5, how one server did not receive any of the 28 sessions. This is possible in a random distribution.

![Figure 6. Distribution of sessions by the servers using a weighted random load balancing approach.](image)

Since it is hard to evaluate the correction of the algorithm simply through the visualization of the results, we will simulate again the establishment of 2000 sessions (results in Figure 6).

Of course, the fraction of sessions established with each server \(i\) should be given by Equation (1),

\[
f_i = \frac{w_i}{\left(\sum_{j=1}^{n} w_j\right)}
\]

where \(w_i\) is the weight attributed to each server. In fact, after running the simulation, we obtain fractions that are similar to what was expected. As it was also expected, the fractions sum to 1.

5 Conclusion

AS Internetworking is a well studied problem whose spectrum is mainly dominated by BGP. The correctness and ubiquity of BGP make it the most popular choice among network administrators. Nevertheless, the push for more flexible inter-AS communication makes SDN-based solutions increasing attractive.

SDN and inter-domain routing have interesting synergies that are worth exploring. Using IXPs to demonstrate the power of SDN is an interesting step towards the proliferation of SDN technology in large scale networks.

Showing how easy it was to implement a new functionality is an indicator of the added power SDN technology gives to network operators.

As with any technology, the way to make SDN-based Internetworking solutions attractive is to make the customers want it more than the previous solutions. The fast development times and low development costs of new SDN solutions make it attractive, as there are no significant downsides to adopting this solution. iSDX, in particular, is fully backwards compatible with non SDN-enabled networks; and even without any SDN capable hardware, a customer may benefit immensely from the usage of inbound and outbound rules. A system like iSDX should offer at least the same quality of service as the previous services.

### References