ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Philipp Mao

Boosting the Convergence Performance of SDX Platforms

Semester Thesis SA-2016-69
October 2016 to January 2017

Tutor: Laurent Vanbever
Supervisor: Rüdiger Birkner, Thomas Holterbach

Abstract

(**RB: after you are done writing all the other parts, make sure that the abstract fits the rest of the thesis**)
Internet outages in BGP are critical and lead to long convergence times .
Industrial-Scale Software defined Internet exchange Points in short iSDX, suffer like any other BGP speaking router from this problem. This is made even worse because of the additional overhead induced by the iSDX's participant policies. Existing fast reroute frameworks like Swift can help shorten convergence times. In this work we implement Swift into the iSDX. Swift can be easily implemented into the iSDX due their similar architecture.
Without a lot of additional overhead, the convergence time of the iSDX is improved by up to a factor 60. But as both Swift and the iSDX use the destination mac address as a data plane tag the amount of bits available to encode information for the iSDX and Swift is halved. This reduces the iSDX's ability to scale with higher number of participants.

Contents

1 Introduction **9**

2 Background **11**
2.1 BGP . . . 11
2.2 iSDX . . . 11
2.2.1 iSDX Architecture . . . 12
2.2.2 Policies . . . 12
2.2.3 Virtual Next-Hop, Virtual MAC Address . . . 12
2.3 Swift . . . 13
2.3.1 Architecture . . . 14
2.3.2 Burst Prediction Algorithm . . . 14
2.3.3 Encoding of Routing Information . . . 15

3 Motivation **17**

4 Implementation **19**
4.1 Architecture . . . 19
4.2 Swift-BPA . . . 20
4.3 FR-Handler . . . 20
4.4 VMAC Partitioning . . . 21
4.5 Changes to the iSDX . . . 21

5 Evaluation **23**
5.1 Test Setup . . . 23
5.2 Convergence Time without Swift . . . 23
5.3 Convergence Time with Swift . . . 24
5.4 Swift Overhead . . . 25

6 Discussion **27**
6.1 Convergence Performance . . . 27
6.2 VMAC Evaluation . . . 27
6.3 Fast Reroute Flow Rules . . . 28
6.3.1 Number of Flow Rules . . . 28
6.3.2 Outbound Policies and Fast Reroute Rules . . . 28

7 Conclusion **31**

List of Figures

2.1 iSDX architecture 12
2.2 Example of outbound and inbound policies at an iSDX connected to three participants 13
2.3 Example of the VMAC at an iSDX connected to three participants 14
2.4 Example topology of a swifted router 14
2.5 Example of a fast reroute after BPA predicts the AS link 300 600 to be down . . . 15
2.6 Example of the Swift vmac 15

4.1 iSDX architecture with Swift 19
4.2 pipeline of the Swift-BPA module 20
4.3 pipeline of the fast reroute handler 20
4.4 example of the VMAC in the iSDX with Swift 21
4.5 pipeline of the modified route server 21

5.1 Test Setup 23
5.2 Convergence time of the iSDX without Swift 24
5.3 Convergence time of the iSDX without Swift 24
5.4 Time spent on detection, prediction and reroute rule installation during a fast reroute 25
5.5 Time to process a single BGP Update with and without Swift 25

6.1 Vmac of the iSDX with Swift 28

List of Tables

Chapter 1

Introduction

(**RB: try to extend the introduction and get a better flow. it reads like multiple fragments. all of a sudden SDN comes into play. Always write Internet with a capital I.**)
The border gateway protocol (BGP) is the glue that holds the Internet together by allowing autonomous networks to exchange information about the reachability of prefixes without revealing their own network infrastructure.
Remote disruptions in BGP can lead to convergence times of multiple minutes. This long convergence time is caused by the way BGP updates are propagated through the Internet. The convergence time is lower bounded by the time it takes for all BGP withdrawal updates to be received.
Software-defined networking allows network operators to have more control over the network routing. It is also a technology that can help solve some of the Internet's problems including long BGP convergence times.
(**RB: why?**)

In this work we show that it is possible to improve the convergence time at software-defined exchange points, such as the industrial scale internet exchange point (iSDX) using the Swift framework. As both the iSDX and Swift rely on a similar infrastructure to enable their functionality, it is a good idea to integrate Swift into iSDX.

In the following, We will first provide some background on BGP, iSDX and Swift in 2. Then we describe the motivation behind implementing Swift in the iSDX in 3. In 4 we evaluate the system both in terms of convergence time and introduced overhead and discuss the results in 5.

Chapter 2

Background

In this chapter, we give a brief overview of BGP in 2.1 and explain the problem of long convergence time upon remote failure in BGP. We highlight the architecture and key insights of both iSDX and Swift in 2.2 and 2.3, respectively. The architecture of both frameworks are very similar, they are based on an SDN switch, an SDN controller and a route server. In the next chapter, we explain how these similarities can be put to use in the implementation.

2.1 BGP

The Border Gateway Protocol (BGP) allows autonomous systems to exchange routing information with each other. The routing information is communicated on a per prefix basis using BGP updates. BGP updates contain attributes. Attributes are the prefix, the next hop and the AS-path the packet will traverse, if packets are sent via this route. There are two types of updates, announcements and withdraws. Announcements inform that the prefix can be reached via this route and withdraws inform that the previously announced prefix cannot be reached via this route anymore.
Upon receiving a BGP update routers make a routing decision using the attributes of the BGP update and then send updates to their peers. Routers only send announcements with their own best route to the prefix and if the routers do not know a route to the prefix they send withdraws.
Convergence time in BGP upon remote failure can be slow. This is because of the way updates propagate trough the network. Changes are only passed on once routers have finished computing the best route to the prefix. Only once all the best path computation has finished, a router is able to make sure that packets do no get sent into a black hole or loop anymore.

2.2 iSDX

The iSDX is an internet exchange point enhanced with an SDN switch and SDN controller. An internet exchange point (IXP) is a physical location where multiple autonomous systems meet to exchange traffic and BGP routes. An exchange point provides a fabric for the networks to interconnect. Each network attaches with one or multiple routers to this fabric. To reduce the burden on the network operators, internet exchange points often provide a route server. Thanks to the route server, each border router only has to peer with the route server instead of all other participants. This allow the participant to receive BGP updates from all other participants. However, even though many different routes might be available for a single prefix, participants of a traditional IXP can only use a single route per prefix.
The iSDX allows its participants to not only make use of these additional routes, but also gives the participants more fine-grained control over the routing decisions. A Participant connects to the iSDX as if it were a traditional IXP and is able to specify special policies that allow to use multiple routes at once and control routing beyond the prefix.
We first show the iSDX architecture in 2.2.1, explain the different policies and their capabilities in 2.2.2 and how the iSDX features are implemented using a virtual next-hop and repurposing the destination mac address in 2.2.3.

2.2.1 iSDX Architecture

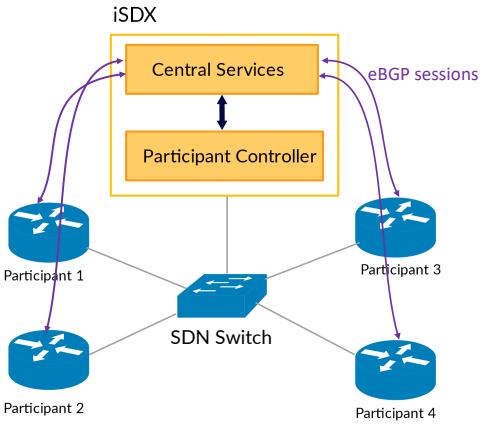


Figure 2.1: iSDX architecture

The iSDX architecture consists of two main parts: *(i)* the Central Services and *(ii)* the Participant Controller.
The Central Services has two tasks. *(i)* It collects all BGP updates and ARP queries centrally and forwards them to the corresponding participant controller. *(ii)* It makes sure that BGP connection, default forwarding and ARP traffic is correctly handled by installing flow rules.
Every participant has its own participant controller. This allows each participant to make its own best path decision. The participant controller receives and processes BGP updates from the central route server. Every participant controller has a local routing information base, which keeps track of all the received routes, currently used routes and routes advertised to other participants. It takes care of the virtual next-hop and VMAC assignment. It handles ARP requests and sends out gratuitous ARP replies. Before BGP updates get sent to the participants border router they are processed by the participant controller. Also, it installs all the flow rules which are necessary for its policies to be enforced.

2.2.2 Policies

Policies allow a participant to specify forwarding behavior which deviates from the default best path forwarding. They can be specified on any part of the header. Policies are implemented as flow rules that the participants controller programs into the SDN switch. Two types of policies exist: *(i)* outbound policies that handle all out-going traffic and *(ii)* inbound policies that handle all incoming traffic.

Outbound Policies: Outbound policies let participants direct packets going from themselves to the iSDX. They allow the participants to forward packets onto another path than the best path. An example of an outbound policy is shown in Figure 2.2, where participant *A* has defined two outbound policies *X* and *Y* directing traffic away from the best path to either one of the other two participants.

Inbound Policies: Inbound policies allow participants to specify how incoming packets are handled. In effect they allow the participant to choose to which of his own routers packets from the iSDX get sent to. An example of an inbound policy is shown in Figure 2.2, where participant C has defined two inbound policies *P* and *Q* directing traffic to either one of its routers depending on the destination port.

2.2.3 Virtual Next-Hop, Virtual MAC Address

Note that when specifying a policy, the network operator does not have to take into account whether the specified destination is actually able to handle that traffic. Some policies might

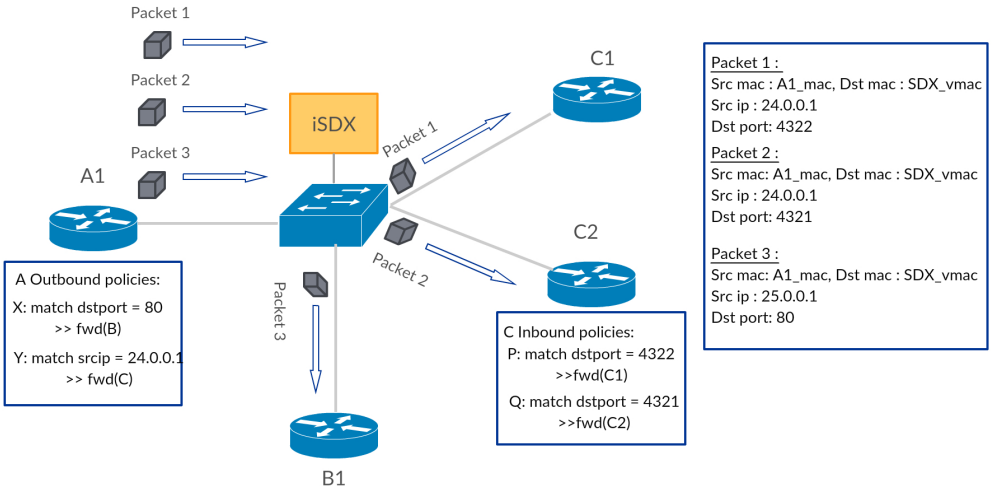


Figure 2.2: Example of outbound and inbound policies at an iSDX connected to three participants

direct packets to participants that did not advertise the prefix to that participant or simply do not know a route to the prefix. Participants are not restricted when defining outbound policies. This is a problem because outbound policies can end up violating BGP advertisements. It is not possible to require participants to only specify feasible policies as the routing information is quite volatile and network operators would constantly need to update their policies to make sure none of the traffic is sent to a black hole.

The iSDX solves this problem by attaching additional information to each packet. This additional information is embedded into the destination mac address, transforming the destination mac addresses of packets traversing the SDN switch into a Virtual Mac Address (VMAC). In the VMAC the participant controller encodes the participants advertising the prefix of the packet and the BGP best next hop participant for this prefix. The first part is used every time a outbound policy is applied. The outbound policy checks if the participant the outbound policy is sending packets to has advertised the prefix. The second part is used if the packet does not match any outbound policy. Default rules in the SDN switch match on the best next-hop participant and send the packet to this participant.

Each border router replaces the destination MAC address with the corresponding VMAC before sending a packet to the iSDX fabric. It is possible to assign this task to the participant's border routers using the next hop attribute in the BGP announcement. Every prefix is assigned a virtual next hop (VNH) that maps to a VMAC. Before sending BGP updates to its border routers the participant controller sets the next hop attribute to the VNH corresponding to the prefix of the update. When a packet arrives at the border router it checks its routing table and finds the next hop, in this case the VNH. The border routers learn the VMAC of the VNHs via ARP. The border router broadcasts a ARP request for the VNH which is received by the ARP proxy, forwarded to the corresponding participant controller and replied to with the corresponding VMAC.

Figure 2.3 shows the VNHs and VMACs used by participant A.

2.3 Swift

Swift is a prediction and fast reroute framework which improves the convergence time of a BGP speaking router upon remote failures. Swifts prediction relies on the fact that the cause of a burst of withdrawals can be predicted before receiving all the withdrawals using the attributes of the already received withdrawals.

In this section, we first show the Swift architecture in 2.3.1, explain the encoding in 2.3.3 and explain how Swift is able to reduce the convergence time of a router in 2.3.2.

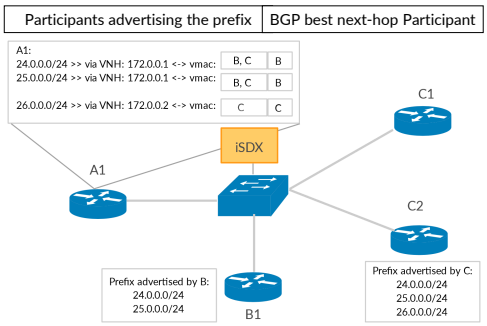


Figure 2.3: Example of the VMAC at an iSDX connected to three participants

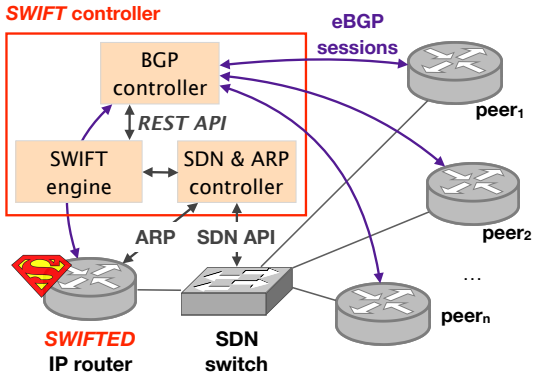


Figure 2.4: Example topology of a swifted router

2.3.1 Architecture

Swift uses an SDN switch connected to the swifted router, its neighbors and to the Swift controller. The Swift Controller has three main parts, *(i)* the BGP controller, *(ii)* the Swift engine and *(iii)* the SDN & ARP controller. The BGP controller receives BGP updates from the peers of the swifted router. The BGP controller forwards the updates to the Swift engine. The Swift engine consists of two main modules: *(i)* the burst prediction algorithm and *(ii)* the encoding of routing information. These two modules are the two main features of Swift. Every peer of the swifted router has its own Swift eninge running. The SDN & ARP controller programs flow rules into the SDN switch and manages ARP requests.
The architecture of Swift is similar to the architecture of the iSDX. Both use a SDN switch connected to multiple BGP speaking routers, the BGP controller like the central services forwards BGP updates to the corresponding Swift engine/participant controller and every participant controller implements the functionality of the SDN & ARP controller.

2.3.2 Burst Prediction Algorithm

The burst prediction algorithm predicts the failed AS-link. It predicts the failed link after detecting a burst. The burst is detected once a certain threshold of withdrawals is reached in a short time frame.
The burst prediction algorithm takes BGP updates. It uses the updates to build an AS-topology. It also stores prefixes and the AS-path of the announcements. Once a burst is triggered the burst prediction algorithm uses the withdrawals, stored AS-paths and the AS-topology to predict the failed AS-link.
Upon predicting a failed link the Swift Controller pushes fast reroute flow rules into the SDN switch matching on the failed AS-link and on the corresponding backup next-hop. In Figure 2.3 the burst prediction module has predicted the AS-link 300 600 to be down. So it pushes rules matching on the AS-path 300 600 and the backup next-hop in this case 400.
Every packet that traverses the failed link (has the failed AS-link in it's AS-path encoding) will get rerouted to the backup neighbor.

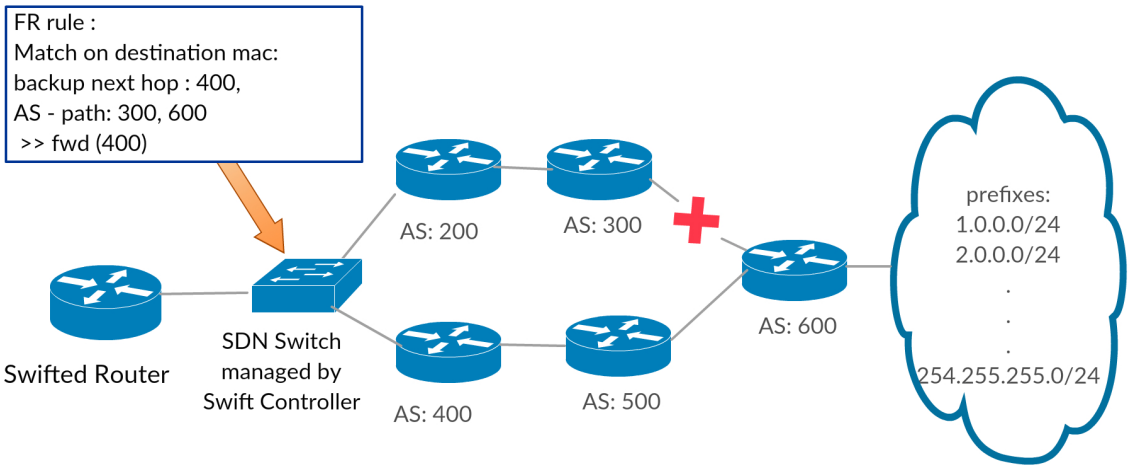


Figure 2.5: Example of a fast reroute after BPA predicts the AS link 300 600 to be down

By predicting the failed link and pushing fast reroute rules instead of waiting for all withdrawals to arrive, Swift reduces the convergence time of the swifted router significantly.

2.3.3 Encoding of Routing Information

Swift similarly to the iSDX uses virtual next hops and the destination mac address to encode the necessary information about the packet and its backup paths.
For every prefix Swift encodes the AS-path up to a certain depth and the backup next hops for each AS-link on that AS-path. Backup next hops are neighbors of the swifted router which also advertise the prefix and their advertised route does not traverse the specific AS-link. This encoding is then mapped to a virtual next hop. When the swifted router wants to send a packet to any prefix it will use the VNH assigned by Swift. The VNH directly maps to the destination MAC address.
Figure 2.6 shows the swifted router and the VMACs used for the prefixes advertised by the neighbors.

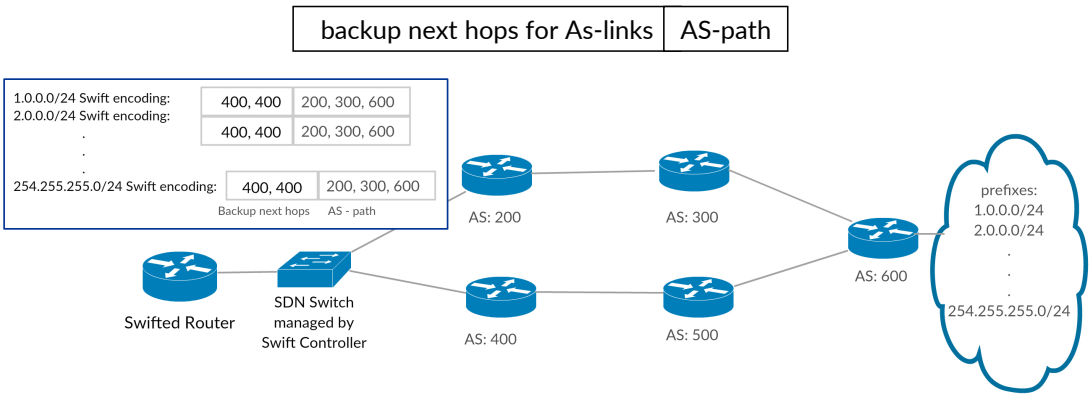


Figure 2.6: Example of the Swift vmac

Chapter 3

Motivation

Routers using BGP can suffer from long convergence times after a remote failure. This also applies to routers connected to an iSDX. At an iSDX implementing Swift once promises to improve the convergence time of all the routers connected to the iSDX. Convergence time being the time from the failure to occur to the packets being sent to the correct participant and not to a black hole. The idea being that Swift can push fast reroute flow rules into the SDN switch and redirect packets to backup participants.
The main reason why implementing Swift in the iSDX is reasonable is the similarity of their architecture: Both the iSDX and Swift use an SDN switch to steer traffic. They are both connected to BGP speaking routers and receive BGP updates from them. Swift and iSDX both use the destination mac address to encode information about a prefix and use the next hop to map this VMAC to a prefix. In addition the Swift framework allows multiple swifted routers to be connected to the SDN switch, which in the iSDX's case means that all the participants will benefit from Swift.
The main challenge when implementing Swift into the iSDX is to change as little as possible in both systems. The iSDX with Swift should implement the full functionality of both the iSDX and Swift. At the same time the overhead added by Swift should not be too big.
In the next chapter, we explain how Swift was implemented in the iSDX.

Chapter 4

Implementation

In this chapter, we show how Swift was implemented into iSDX. We give an overview of the modified iSDX architecture in 4.1. We explain the two new modules that were added to the iSDX in 4.2 and in 4.3, respectively. We explain the VMAC partitioning in 4.4 and explain what was changed in the default iSDX modules in 4.5.
In the next chapter we present the results of the tests done to measure the convergence performance of the iSDX.

4.1 Architecture

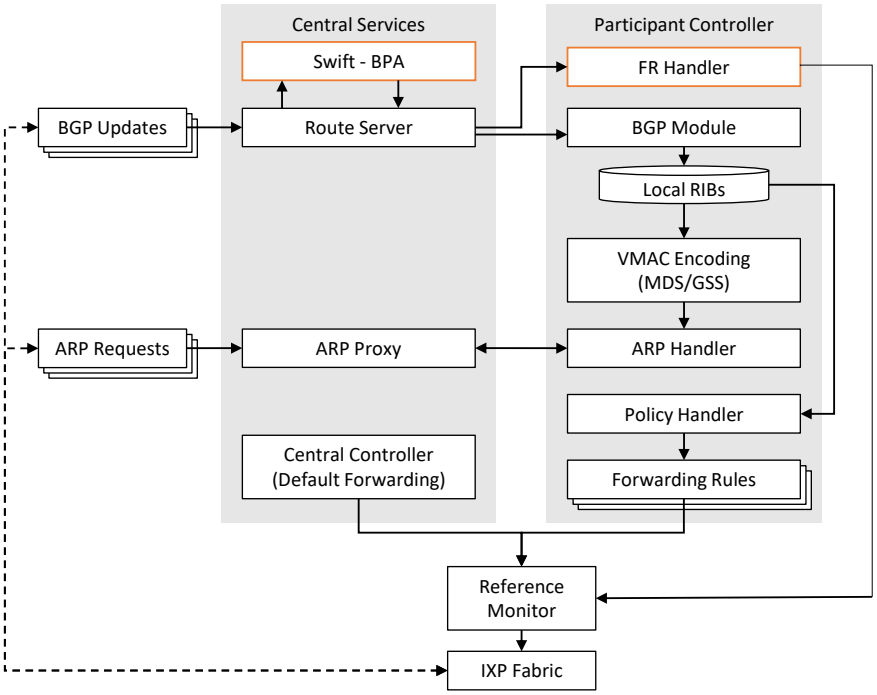


Figure 4.1: iSDX architecture with Swift

Figure 4.1 shows the iSDX [4] architecture with Swift. The orange modules represent the new modules we had to add to the iSDX to implement Swift. The iSDX receives two additional modules the Swift-BPA module in the central services and the FR handler in the participant controller. With these two modules the iSDX will now detect bursts of withdrawals, predict the failed AS-link and push fast reroute rules into the IXP fabric.
In the following sections, I will go over the functionality of the new modules and other changes to the iSDX in more detail.

4.2 Swift-BPA

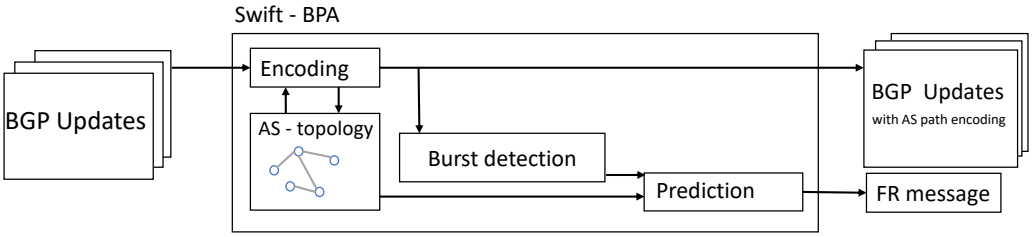


Figure 4.2: pipeline of the Swift-BPA module

The Swift-BPA module implements Swifts main functionality, encoding and prediction. It is part of the central services and exchanges BGP updates and FR messages with the route server and participant controllers.
The Swift-BPA module is placed at the route server and processes each update received by the route server before passing it on to the participant controllers. When receiving a BGP update the Swift-BPA updates the AS-topology and adds the AS-path encoding to the BGP update. Then the update gets sent to the participant controller. After the update is sent to the participant controller the burst detection checks if the update triggers a burst. If so the prediction module predicts the failed AS-link and sends FR messages to the participant controller. The fast reroute message informs the participant controllers about the AS-link that is predicted to be down.
Similar to the participant controller every participant has a Swift-BPA process running. The Swift-BPA only receives BGP updates from his own routers. This way the Swift engine can be implemented without any modifications.

4.3 FR-Handler

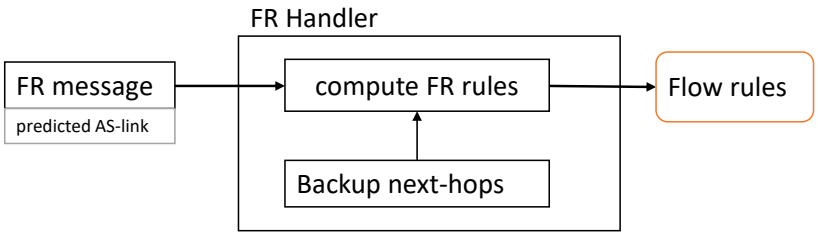


Figure 4.3: pipeline of the fast reroute handler

The FR-handler implements the pushing of fast reroute rules.
Once the Swift-BPA predicts a failed AS-link it sends FR messages to all the participant controllers. The FR-handler receives FR messages and computes the fast reroute rules using the backup next-hops and the predicted AS-link. Every participant controller computes its own backup next-hops.
The fast reroute rules get sent to the reference monitor as flow rule messages. The reference monitor then installs the flow rules in the SDN switch. Just like in Swift the fast reroute rules match on the failed AS-link and on the backup next-hop.

4.4 VMAC Partitioning

Both Swift and the iSDX use the destination mac address as a VMAC to attach additional information to the packet. It is not easy to use another field of the header as only the destination mac address can be changed by modifying the next hop attribute. Therefore the VMAC has to be shared between the iSDX and the Swift encoding. The number of bits available to encode information for the iSDX and Swift is reduced. This is because the iSDX and Swift encode different information about the prefix. The iSDX encodes the participants advertising the prefix and the BGP best next hop. Swift encodes the AS-path and the backup next hops for each link on the AS-path.
The encoded AS-path starts with the second AS on the AS-path. This is because the first AS is already encoded as the BGP best next-hop. (in the iSDX part of the VMAC)

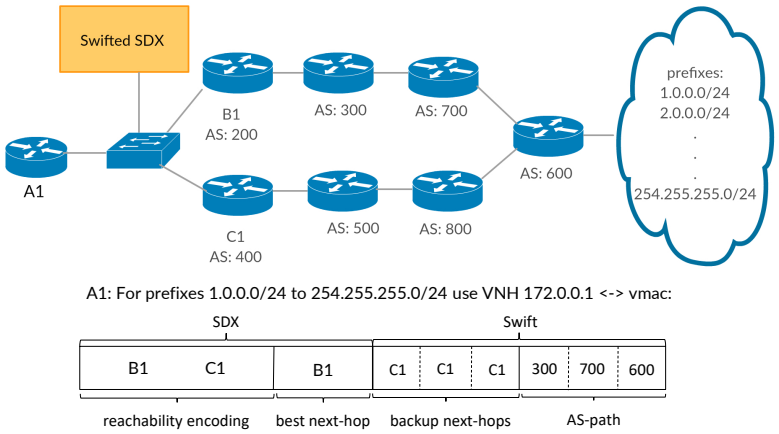


Figure 4.4: example of the VMAC in the iSDX with Swift

4.5 Changes to the iSDX

We also had to apply a few changes to the iSDX's modules. These changes mainly affect the route server, the local RIB of the participant controller and the VMAC encoding.

Route Server: The route server had to be adapted to work with the Swift-BPA module. The route server now has two modules the Listener and Sender.
The Listener receives BGP updates and forwards them to the Swift-BPA.
The Sender receives modified BGP updates and FR messages from the Swift-BPA and forwards them to the participant controllers. The sender processes FR messages with higher priority than modified BGP updates, this way the FR messages reach the FR handlers as quickly as possible.

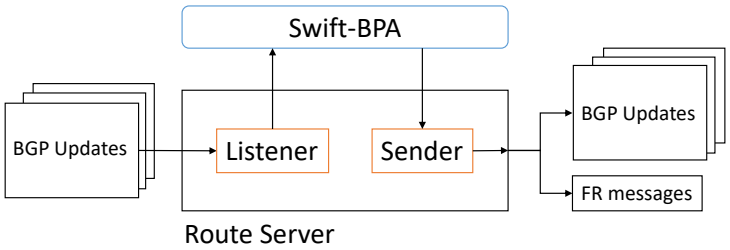


Figure 4.5: pipeline of the modified route server

Local RIB: The local RIB now stores the AS-path encoding. When processing a BGP update the local RIB extracts the AS-path encoding (added by the Swift-BPA) and saves it as an additional attribute. The encoded AS-path is then used in the VMAC encoding.

VMAC Encoding: The VMAC encoding now computes the backup next-hops for the AS-path of the BGP update. There is one backup next-hop for every AS-link on the AS-path, packets can be sent to the backup next-hop in case this AS-link is predicted to be down.
The VMAC encoding now builds the VMAC using both iSDX and Swift encoding.

Chapter 5

Evaluation

In this chapter, we present the results of the tests we ran to evaluate the convergence performance of the iSDX. We explain the test setup in 5.1. We present the convergence time of the iSDX without Swift in 5.2, the convergence time of the iSDX with Swift in 5.3 and examine the Swift overhead in 5.4.
In the next chapter we discuss the results, examine the VMAC partitioning and the fast reroute rules.

5.1 Test Setup

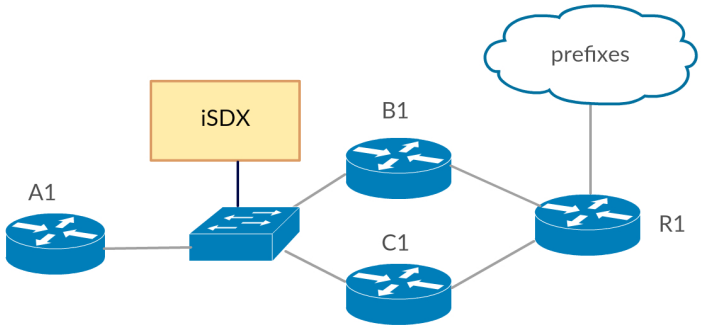
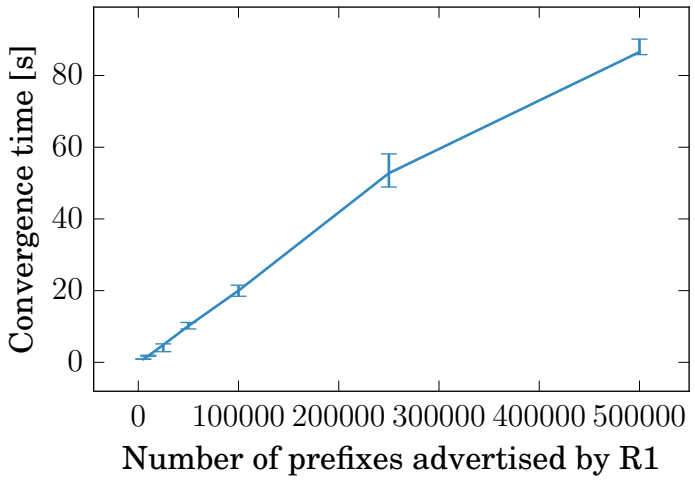


Figure 5.1: Test Setup

Figure 5.1 shows the test setup. The test setup has an iSDX with or without Swift connected to three participants. Participants B1 and C1 are connected to the rest of the internet via R1 and advertise up to 500000 prefixes to A1. Participant A1 prefers routes from B1. Remote failure is simulated by setting the link B1 R1 down. If this link is down A1 needs to updates his RIB, check if flow rules have changed and update the virtual next-hop/vmac for every withdrawn prefix.
The experiment setup is run on a server with following specs: Intel Xeon CPU E5620 with 4 Cores at 2.4 GHz, 36GB of RAM, running Ubuntu 14.04. The experiment uses Mininet [2] to simulate the network. The routers A1, B1, C1 and R1 are quagga [3] routers. The perl script bgpsimple [1] is used to inject an arbitrary number of routes into R1.

5.2 Convergence Time without Swift

We measure the convergence time as the time between the first withdraw arriving at the route server and the participant controller finishing to process the last withdraw. To measure the convergence time we use the built in iSDX log server.
This convergence time does not take into account the hold timer or the time the participant router takes to process the withdrawals. But since these things are not under the control of the iSDX they are ignored in this evaluation.



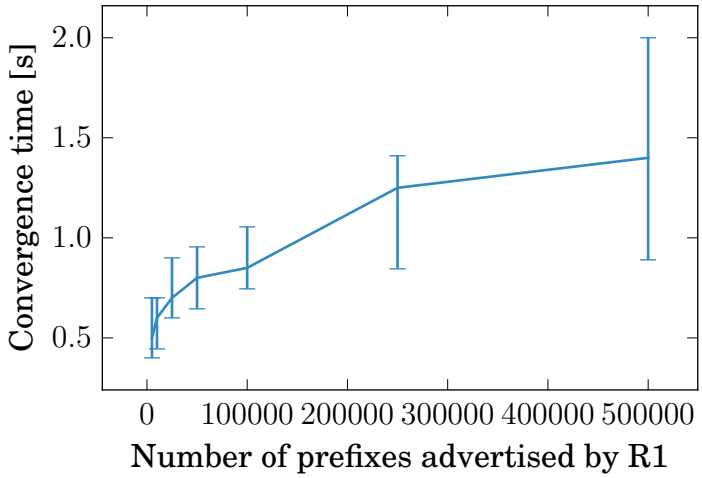
prefixes	Conv. time
5K	0.9 s
10K	1.8 s
25K	4.9 s
50K	10.1 s
100K	20 s
250K	52.75 s
500K	86.5 s

Figure 5.2: Convergence time of the iSDX without Swift

Figure 5.2 shows the convergence time depending on the number of prefixes injected by bgpsimple. The results show that the convergence time increases linearly with the number of prefixes advertised by R1.
At 500'000 prefixes the iSDX takes about 90 seconds to converge. During these 90 seconds A1 is still sending packet to B1 even though that route does not exist anymore. Hence, all the traffic is dropped.

5.3 Convergence Time with Swift

Here, we measure the convergence time as the time between the first withdraw arriving at the route server and the participant controller's FR-handler finishing to push the Fast-reroute rules.



prefixes	Conv. time
5K	0.5 s
10K	0.6 s
25K	0.7 s
50K	0.8 s
100K	0.85 s
250K	1.25 s
500K	1.4 s

Figure 5.3: Convergence time of the iSDX without Swift

Figure 5.3 shows the convergence time depending on the number of prefixes injected by bgpsimple. The convergence time increases slightly with higher number of prefixes. At 500'000 prefixes the iSDX takes about 1.5 seconds to push the FR rules. After these 1.5 seconds packets sent from A1 get redirected to C1 and reach their destination.

Figure 5.4 shows the fraction of the total convergence time and what this time is made up of. In blue is the time until a burst of withdrawals is detected. In red is the time it takes to predict the

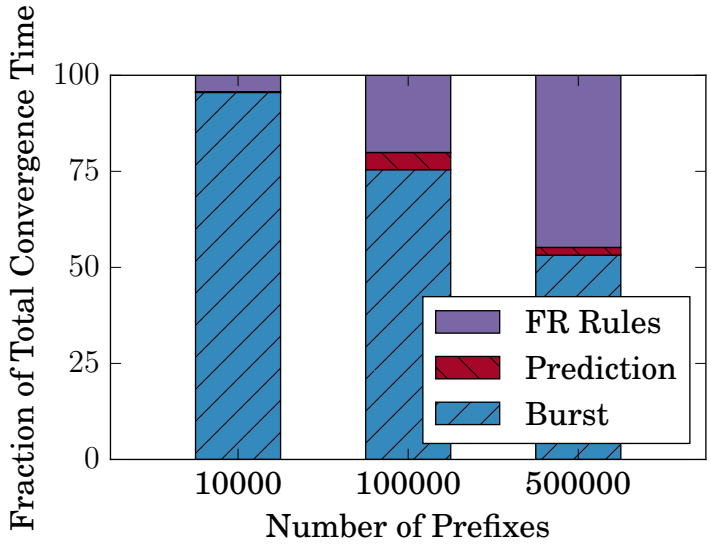


Figure 5.4: Time spent on detection, prediction and reroute rule installation during a fast reroute

failed AS-link. In violet is the time it takes for the participant controller to receive and push the Fast Reroute rules. With a higher number of prefixes withdrawn rises the time until Fast Reroute rules are pushed. The time for the prediction increases slightly with a higher number of prefixes. The time to detect the burst is more or less constant since we used the same Swift configuration for all the experiments.
(**RB: you will most certainly get a question why it takes so much longer to install the fast reroute flow rules for 500k prefixes. Make sure to look this up and also write this in the thesis.**)

5.4 Swift Overhead

In this section we present the overhead that Swift adds to the processing of a single BGP update. We measure the time between the BGP update arriving in the route server and the participant controller finishing to process the update.

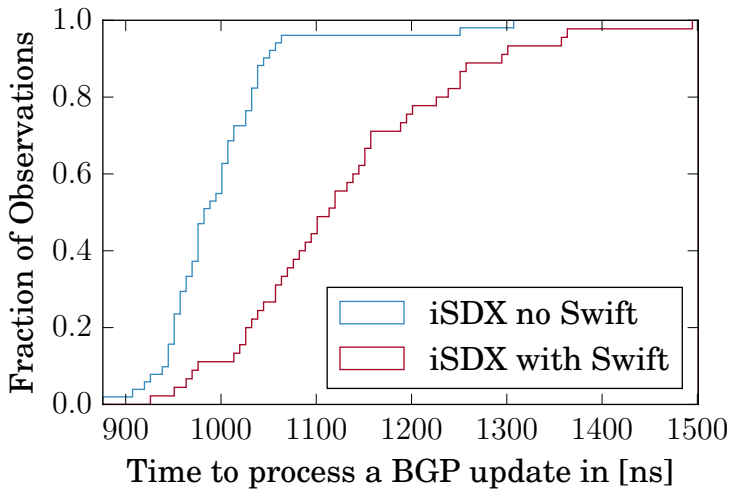


Figure 5.5: Time to process a single BGP Update with and without Swift

Figure 5.5 shows the time to process a single BGP. On average iSDX takes 987ns without Swift

and 1119ns with Swift. We see that Swift increases the processing time by a factor 1.13.

Chapter 6

Discussion

(**RB: this chapter needs still some polishing.**)

In this chapter, we discuss the results of the previous chapter 6.1. We evaluate the VMAC partitioning in 6.2 and examine the number of required fast reroute flow rules in 6.3.

6.1 Convergence Performance

In this section will discuss the results of the measurements presented in 5.
As clearly seen in Figure 5.2 and Figure 5.3 iSDX with Swift has a much shorter convergence time. At 500'000 prefixes the convergence time is improved by a factor of 60. The convergence time of iSDX with Swift increases slightly with a higher number of prefixes. As seen in Figure 5.4 with more prefixes more time is spent until the Fast Reroute rules are pushed into the SDN switch. We suppose that this is because the participant controller has to process more BGP updates before it can process the FR message, as the FR messages and BGP udpates get sent over the same channel.
The shorter convergence time comes at a cost. That is the overhead Swift adds to the processing of a single BGP update. Since the updates get sent to Swift-BPA module before getting sent to the participant controller it takes iSDX with Swift longer to process a single BGP update. On average it takes iSDX with Swift about 132ns longer to process a single BGP update, 13 % longer.

6.2 VMAC Evaluation

Since both the iSDX and Swift use the destination mac address to encode the required additional information, the amount of bits available to the iSDX and Swift encoding is reduced as the space has to be shared. The current VMAC partitioning is simply the first intuition on how the VMAC can be partitioned. The partitioning can be easily changed by configuring the iSDXs parameters. There may still be some optimization. But we realize that optimizing the VMAC partitioning would outgrow the time frame of this project. In this section we evaluate the VMAC partitioning in its current state.

(**RB: refer to figure**)
In its current state 24 bits are allocated to the iSDX. 16 bits for the reachability encoding and 8 bits for the BGP best next-hop.
The amount of bits allocated for the best next-hop limits the number of participants the iSDX with Swift can have. The number of participants is limited to 2^8 = 256.
24 bits are allocated to Swift. 12 bits are used to encode backup next-hops and 12 bits are used for the AS-path encoding.
With 12 bits used for the AS-path encoding the encoding has a coverage performance of about 85%. (see Swift paper Figure 9)
(**RB: add proper citation**)

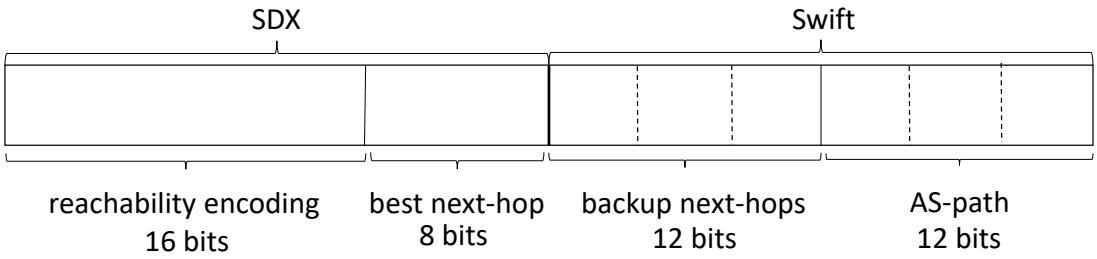


Figure 6.1: Vmac of the iSDX with Swift

12 bits for 3 backup next-hops means 4 bits for each next-hop. This means that every participant can have 16 backup next-hops at most. This is not a lot and after 16 backup next-hops have been assigned some prefixes will end up with no backup next-hop. On the other hand, it also limits the number of fast reroute rules.

6.3 Fast Reroute Flow Rules

In this section we will first examine the number of Fast reroute Flow Rules required for a Fast-Reroute in **??**. We will also examine the priority of the Fast-Reroute Flow Rules compared to Outbound Policies in **??**

6.3.1 Number of Flow Rules

(RB: important to emphasize that on what the number of fast reroute flow rules depends. It does not depend on the number of withdrawn prefixes.)

After a Fast Reroute message has been received the FR handler pushes Fast Reroute rules into the IXP fabric. For every backup next-hop that the participant has stored a rule is pushed. This means the maximum number of rules pushed by a participant after a Fast Reroute is 16.
FR messages get sent to every participant that is peering with the participant whose Swift-BPA triggered the Fast Reroute. This means the maximum number of flow rules pushed after a Fast Reroute is triggered is 256*16 = 4096. This amount of flow rules is not substantial enough to have a significant impact on the iSDX. With the number of participants limited to 256 and participants having a reasonbale number policies, the flow rule limit for current SDN switches should not be reached. (iSDX paper figure 3 (a) amsix paper figure 9)
(RB: proper citation)

Usually this upper bound of flow rules should not be reached. Upon a Fast-Reroute not all participants will be peering with the participant that triggered the Fast Reroute. Also the participants may not have reached the maximum number of Flow Rules yet.

6.3.2 Outbound Policies and Fast Reroute Rules

(RB: improve the writing. it is not quite clear what the higher priorities actually mean)
Fast Reroute rules and Outbound Policies are in conflict.
If Outbound policies have a higher priority than Fast Reroute rules, packets that match the policy may be rerouted to a backup next-hop or they may be rerouted to a participant whose BGP route traverses the failed AS-link.
If Fast Reroute rules have a higher priority then the participants Outbound policies will be ignored.
(RB: are ignored in the case of a fast reroute even though the policy is not affected by

the outbound policy.)

Due to the limited number of bits available, encoding the AS-path and backup next-hops of the participants routes in the reachability encoding is not feasible. In the current iSDX with Swift Fast Reroute rules have a higher priority than Outbound policies. Future work may allow participants to define which Outbound policies they want overridden by Swift and which ones not. But this goes beyond the scope of this project.

Chapter 7

Conclusion

In this project Swift was implemented into the iSDX without significantly changing either iSDX or Swift.
The convergence time of the iSDX was significantly reduced without too many additional flow rules. Their similar architecture makes integrating one into the other easy but it also severely constrains the iSDX's ability to scale with a higher number of participants. This is due to the bottleneck created by the limited size of the destination MAC address. With the current design up to 256 participants can be supported. At the point of this work only 1.8% of all IXP's have more than 256 participants. (www.pch.net/ixp/dir)
(**RB: citation**)

If the swifted iSDX should be deployed at an IXP with more participants, more bits need to be allocated to the iSDX part of the vmac. This in turn impacts the performance of Swift, leading to traffic being unnecessarily redirected or failed links not being correctly detected. One might look into implementing a more lightweight fast reroute framework that does not need to encode information on the destination mac address.
(**RB: mention future work: investigate optimal swift isdx VMAC partitioning, etc.**)

Bibliography

[1] Bgpsimple simple bgp peering and route injection script. https://github.com/xdel/bgpsimple. Accessed: 2016-11-02.

[2] Mininet an instant virtual network on your laptop. http://mininet.org/. Accessed: 2017-01-06.

[3] Quagga routing software suite. http://www.nongnu.org/quagga/. Accessed: 2016-10-28.

[4] N. Feamster, J. Rexford, S. Shenker, R. Clark, R. Hutchins, D. Levin, and J. Bailey. SDX: A Software Defined Internet Exchange. *Open Networking Summit*, page 1, 2013.