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

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. 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 overhead the convergence time of all the participants of the iSDX is improved by up to a factor 60. But as both Swift and the iSDX use the destination MAC address to encode information, the amount of bits available to Swift and iSDX is reduced. This impacts Swift performance and the iSDX's ability to scale with a higher number of participants.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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. Standard BGP routers have to update the data plane on a per prefix basis. The router has converged once the last withdrawal has been received and processed. Swift is a fast reroute and prediction framework that can be deployed at a BGP router. It predicts the cause of the remote disruption and fast reroutes traffic, thus shortening the convergence time significantly. Swift is based on software defined networking.
Software defined networking allows network operators to have more control over the network routing. Software defined networking abstracts lower level functionality and makes network control directly programmable. Software defined networking has a lot of applications. In this work we will focus on the industrial scale Internet exchange point (iSDX).
In this project we show that it is possible to improve the convergence time at software defined exchange points, such as the iSDX using the Swift framework. As both the iSDX and Swift rely on a similar infrastructure to enable their functionality, it is intuitive to integrate Swift into the iSDX.
In the following, We will first provide some background on BGP, iSDX and Swift in chapter 2. Then we describe the motivation behind implementing Swift in the iSDX in chapter 3. In chapter 4 we present the design of the iSDX with Swift. In chapter 5 we evaluate the system both in terms of convergence time and introduced overhead and discuss the results in chapter 6.

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 updates have reached a router and it has finished the best path computation for all the updates, has the router converged. Meaning the router is able to make sure packets do not get sent into a loop or to a black hole anymore. The convergence time of a router is lower bounded by the time it takes for all the information/updates to be received. Because all routers before also have to process the updates, the convergence time increases the further away the failure takes place.

2.2 iSDX

The iSDX is an Internet exchange point enhanced with a software defined networking (SDN) switch and a 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 allows participants 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 the participant 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

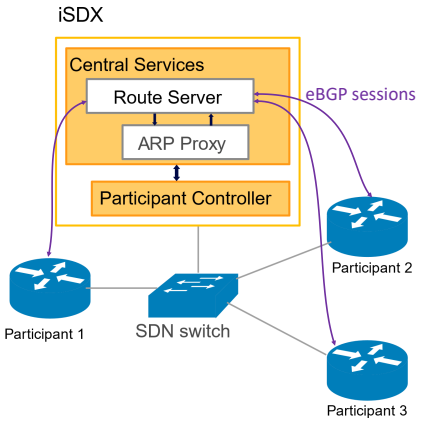


Figure 2.1: iSDX architecture

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 (RIB), 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 (VNH) and virtual MAC address (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. It also 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.

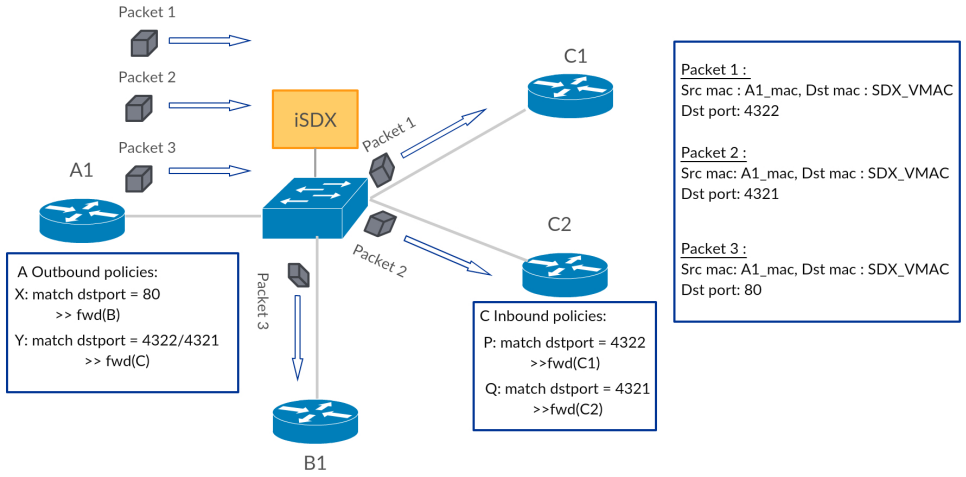


Figure 2.2: Example of outbound and inbound policies at an iSDX connected to three participants

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 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 (reachability) 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.

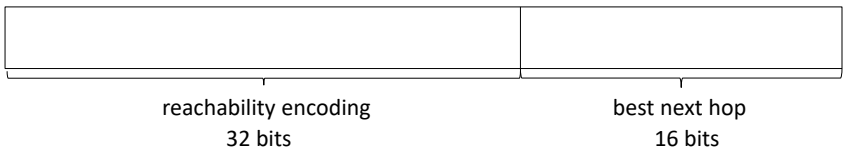


Figure 2.3: VMAC of the iSDX

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 2.4 shows the VNHs and VMACs used by participant A.

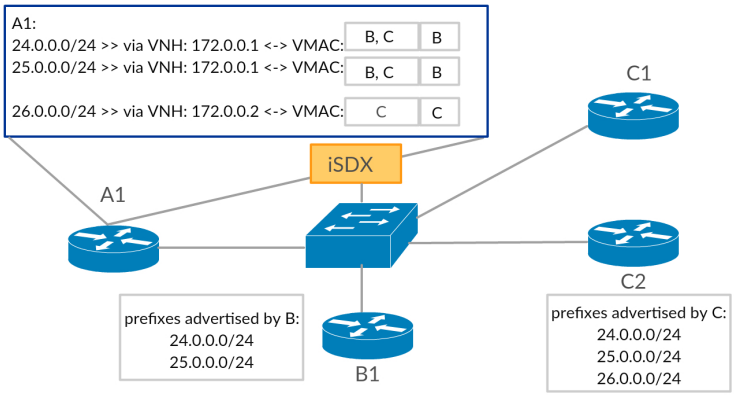


Figure 2.4: Example of the VMAC at an iSDX connected to three participants

2.3 Swift

Swift is a prediction and fast reroute (FR) framework which improves the convergence time of a BGP speaking router upon remote failures. Swift speeds up the learning phase by predicting the cause of the withdrawals. 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. Swift also speeds up the updating of the data plane. After the cause of a burst of withdrawals is detected fast reroute rules are installed into the SDN switch rerouting packets traversing the failed AS-link. The number of fast reroute rules is independent of the number of withdrawn prefixes thus the problem of having to update the data plane for every prefix is avoided.
In this section, we first show the Swift architecture in 2.3.1, explain how Swift is able to reduce the convergence time of a router in 2.3.2 and explain the encoding in 2.3.3.

2.3.1 Architecture

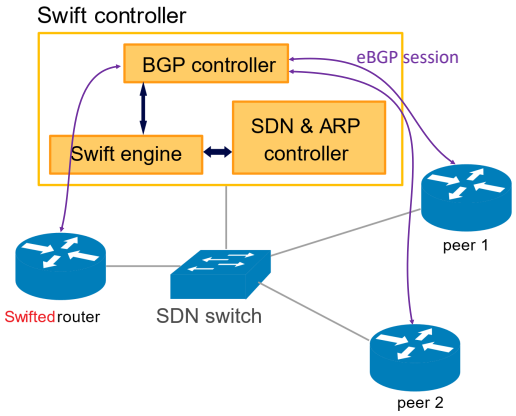


Figure 2.5: Example topology of a swifted router [7]

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 (BPA) 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 burst prediction algorithm 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

Upon remote failure the burst prediction algorithm predicts the failed AS-link. It predicts the failed link after detecting a burst. The burst is detected once the number of received updates in a short time frame reaches a threshold.
The burst prediction algorithm takes BGP updates. It uses the updates to build a AS-topology. It also stores prefixes and the AS-path to reach these prefixes. Once a burst is triggered the burst prediction algorithm uses the received updates, stored AS-paths and the AS-topology to predict the failed AS-link. For every AS-link in the AS-topology the prediction algorithm computes a fitting score (FS). The FS takes into account the number of withdrawn routes traversing this AS-link and the number of still used routes traversing this AS-link. The link with the highest FS is predicted to be down.
Upon predicting a failed link the Swift Controller pushes FR flow rules into the SDN switch matching on the failed AS-link and on the corresponding backup next hop. In Figure 2.6 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.

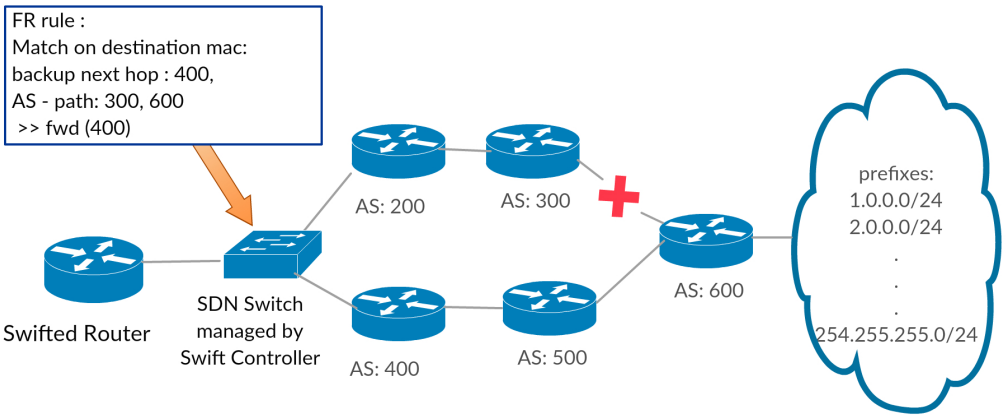


Figure 2.6: Example of a fast reroute

By predicting the failed link and pushing FR 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. If no backup next hop is found Swift simply encodes the second best BGP next hop in place of the backup next hop. This encoding is then mapped to a VNH. 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 VMAC.
Figure 2.8 shows the swifted router and the VMACs used for the prefixes advertised by the neighbors. The Swifted router sends all his traffic via AS 200. AS 400 is the backup next hop for AS-link 200 300 and 300 600. This is because AS 400 advertises the same prefixes as AS 200 and the advertised routes do not traverse the AS-links of the AS-path encoding (the AS-path of the routes advertised by AS 200).

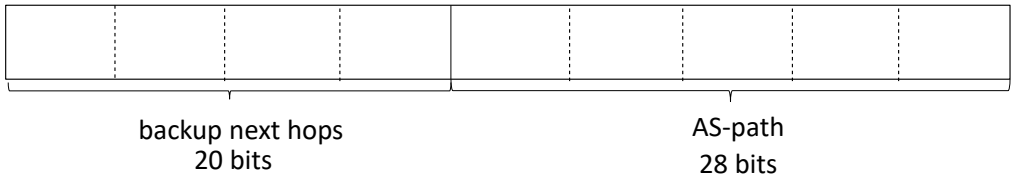


Figure 2.7: Swift VMAC

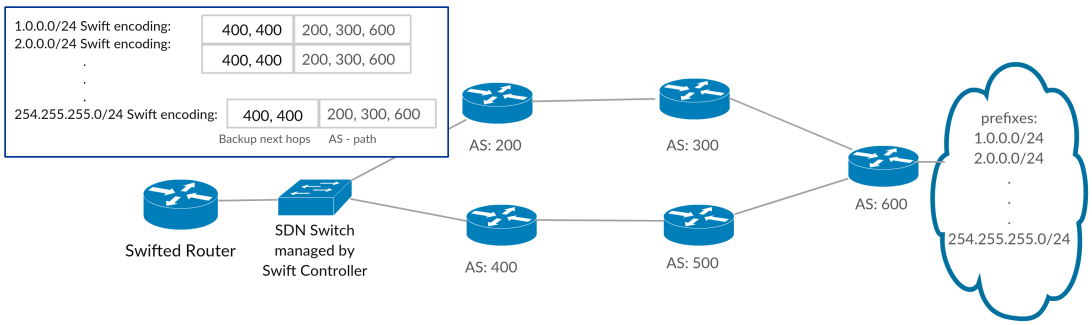


Figure 2.8: Example of the Swift VMAC

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. The idea being that Swift can push FR flow rules into the IXP fabric 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 VMACs 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 we integrated Swift into the 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 4.3. We then show how we combined the VMAC schemes of Swift and iSDX and partitioned the available bits among the two in 4.4. At last, we explain how the default iSDX modules had to be adapted 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

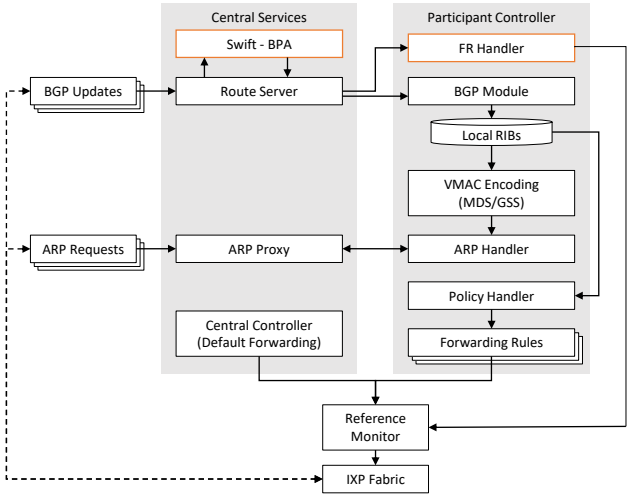


Figure 4.1: iSDX architecture with Swift

Figure 4.1 shows the iSDX [5] 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 is able to detect bursts of withdrawals, predict the failed AS-link that caused the burst and push FR rules into the IXP fabric.

In the following sections, we explain the functionality of the new modules and other changes to the iSDX in more detail.

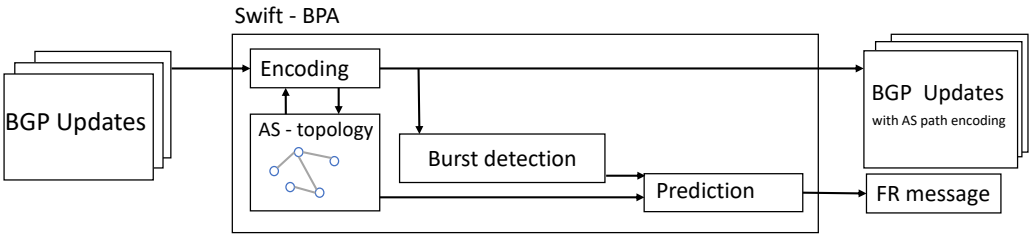


Figure 4.2: Pipeline of the Swift-BPA module

4.2 Swift-BPA

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 in the route server and processes each update received by the route server before passing it on to the participant controllers. Every participant has its own 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.
Figure 4.2 shows the pipeline of the Swift-BPA module. When receiving a BGP update the Swift-BPA updates the AS-topology and adds the AS-path encoding to the BGP update.
The AS-topology is used when predicting a failed AS-link, it stores AS-links and the number of received routes that traverse this link.
The BGP update with the encoded AS-path gets sent to the participant controller. The AS-path encoding is used in the Swift part of the VMAC.
After the update is sent to the participant controller the burst detection checks if the update triggers a fast reroute. If so the prediction module predicts the failed AS-link, the cause of the burst and sends FR messages to the participant controllers.
The FR messages inform the participant controllers about the AS-link that is predicted to be down. In the following, we show how the participant controllers handle the FR messages.

4.3 FR Handler

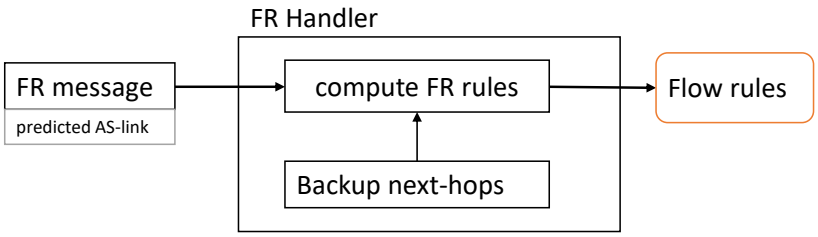


Figure 4.3: Pipeline of the FR handler

The FR-handler is placed in the participant controller and processes FR messages. Upon receiving a FR message the FR handler computes the FR rules. The FR rules are computed using the failed AS-link contained in the FR message and the backup next hops, which were computed by the participant controller. The FR rules get sent to the reference monitor as flow rule messages. The reference monitor then installs the flow rules in the IXP fabric.
The reference monitor is shown in Figure 4.1, it receives flow rule messages from the central services, participant controllers and FR handlers and installs the corresponding rules into the IXP fabric.
Just like in Swift the FR rules match on the failed AS-link and on the backup next hop.

4.4 VMAC Partitioning

Both Swift and 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. 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 4.5 shows an example of a VMAC in a simple topology with a Swifted iSDX connected to three participants.

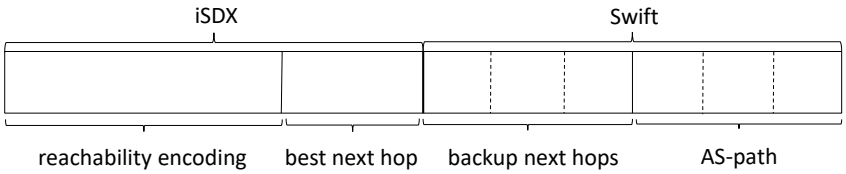


Figure 4.4: VMAC of the iSDX with Swift

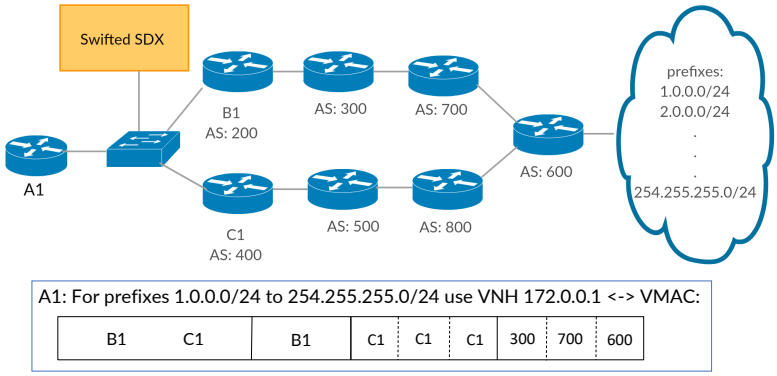


Figure 4.5: Example of the VMAC in the iSDX with Swift

4.5 Changes to the iSDX

We 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. Instead of simply forwarding BGP updates from participant routers to the corresponding participant controller the route server forwards these updates to the corresponding Swift-BPA process. This is done so the Swift-BPA can augment the updates with the encoding of the AS-path. The route server also receives FR messages and augmented BGP updates with AS-path encoding from the Swift-BPA and forwards them to the corresponding participant controllers.

Local RIB: As each BGP update from the route server contains the AS-path encoding, the local RIB had to be adapted to store this information. 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.

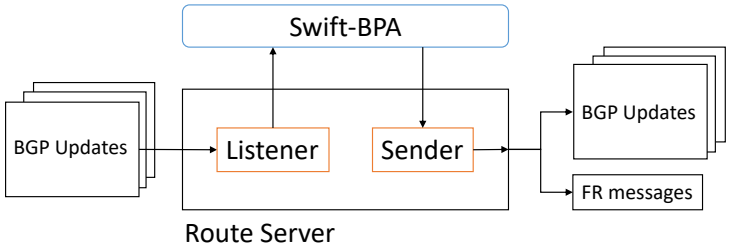


Figure 4.6: Pipeline of the modified route server

VMAC Encoding: The VMAC encoding builds the VMAC with both the iSDX and Swift information. The Swift encoding uses the AS-path encoding stored in the local RIB and the backup next hops. The backup next hops are computed by the VMAC encoding for the encoded AS-paths. 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.

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.
The results show that iSDX with Swift converges about 60 times faster. However, this comes at the cost of a 13% increase in processing time for BGP updates. The majority of the convergence time is spent on detecting the burst.
In the next chapter we discuss the results, examine the VMAC partitioning and the FR rules.

5.1 Test Setup

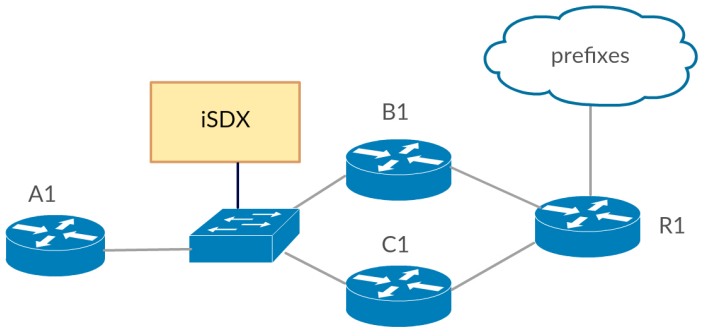
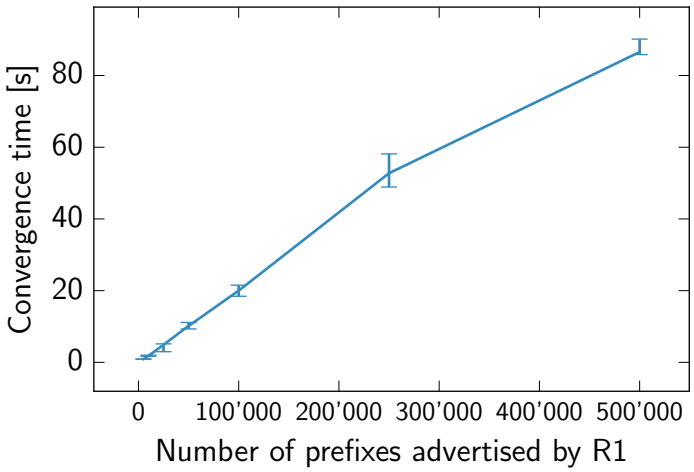


Figure 5.1: Test Setup

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 $500'000$ prefixes to *A1*. Participant *A1* prefers routes from *B1*. Remote failure is simulated by setting the link between *B1* and *R1* down. If this link is down *A1* needs to updates his RIB, check if flow rules have changed and update the VNH/VMAC for every withdrawn prefix.
The experiment setup is run on a server running Ubuntu 14.04 with the following specs: Intel Xeon CPU E5620 with four Cores at 2.4 GHz, 36 GB of RAM. We use Mininet [3] to simulate the network. The routers *A1*, *B1*, *C1* and *R1* are quagga [4] 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.



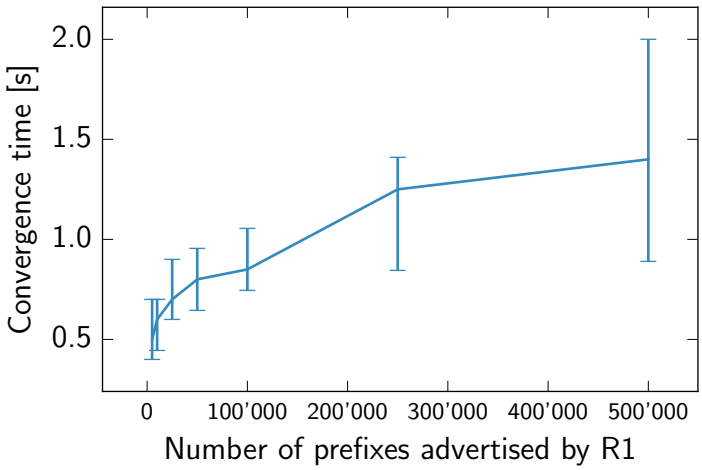
prefixes	Conv. time
5K	0.9 s
10K	1.8 s
25K	4.9 s
50K	10.1 s
100K	20 s
250K	52.75 s
500K	86.5 s

Figure 5.2: Convergence time of the iSDX without Swift

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.
Figure 5.2 shows the convergence time in relation to 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

In this section, 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 FR rules. We use Swifts default parameters, the number of withdrawals needed to trigger a fast reroute is $2'500$.



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 in relation to 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.

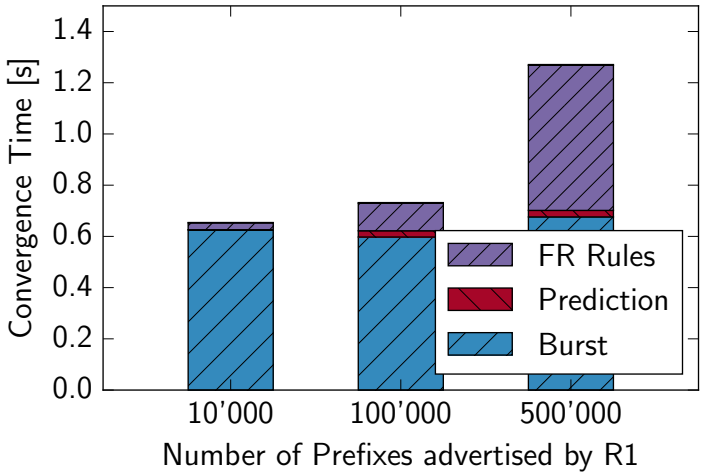


Figure 5.4: Time spent on detection different activities during a fast reroute

Figure 5.4 shows what the convergence 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 failed AS-link. In violet is the time after the Swift-BPA sends FR messages until the participant controller receives and pushes the FR 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.

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. To measure this we use *R1* to advertise a single route and set the link between *B1* and *R1* down and then up. This gives *A1* a single withdrawal and then a single announcement to process.

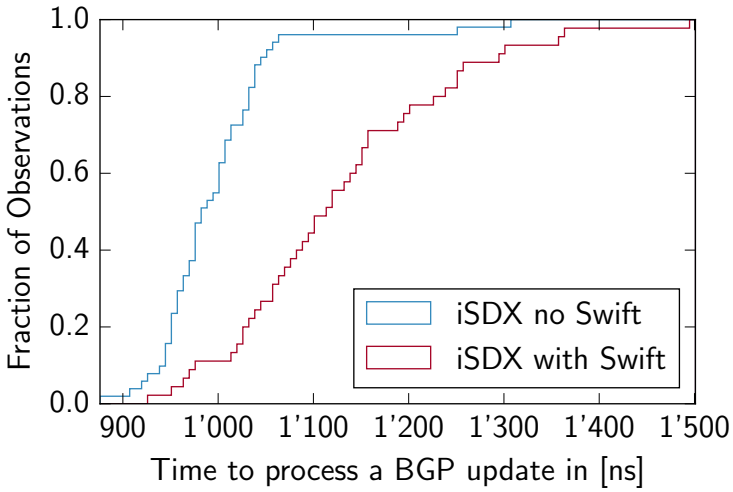


Figure 5.5: Time to process a single BGP Update with and without Swift

Figure 5.5 shows the time to process a single BGP update. On average iSDX takes $987\,\text{ns}$

without Swift and 1119 ns with Swift. We see that Swift increases the processing time by a factor 1.13.

Chapter 6

Discussion

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

6.1 Convergence Performance

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 the 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 FR rules are pushed into the IXP fabric. 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 improvement in the 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 the 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 132 ns longer to process a single BGP update, that is an increase of a factor 1.13.

6.2 VMAC Evaluation

Since both the iSDX and Swift use the destination MAC address to encode 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 easily be changed by adapting the iSDX's configuration parameters. This partitioning is not yet optimal and can certainly be improved. This was not part of this semester thesis. In this section we evaluate the VMAC partitioning in its current state.

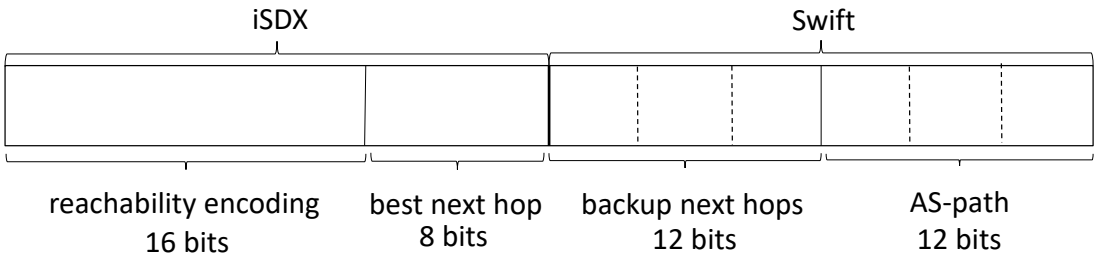


Figure 6.1: Detailed VMAC of the iSDX with Swift

Figure 6.1 show the VMAC partitioning with the number of bits allocated for each part. 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 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%. [7]
12 bits for 3 backup next hops means that for each next hop 4 bits are used. 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 FR rules.

6.3 Fast Reroute Flow Rules

In this section we will first examine the number of FR flow rules required for a fast reroute in 6.3.1. We will also examine the priority of the FR flow rules compared to outbound policies in 6.3.2

6.3.1 Number of Flow Rules

After a FR message has been received FR rules are pushed into the IXP fabric. The number of FR rules does not depend on the number of withdrawn prefixes. It depends on the number of backup next hops and number of participant controllers. In this subsection we examine the number of flow rules required for a fast reroute after a Swift-BPA module detects a failed link. Note that during a burst more than one failed link may be detected.
For every backup next hop that the participant controller has stored a rule is pushed. This means the maximum number of rules pushed by a participant controller after a fast reroute is 16. 16 is the number of backup next-hops available to every participant. (See 6.2)
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*$16 = 4096$. 256 is the maximum number of participants the iSDX with Swift can have. (See 6.2)
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 reasonable number policies, the flow rule limit for current SDN switches should not be reached. [6, Figure 3 (a)]
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 backup next hops yet.

6.3.2 Outbound Policies and Fast Reroute Rules

Flow rules have different priorities. Packets get forwarded depending on the flow rule with the highest priority.
Outbound policies direct packets away from the BGP best path. Swift reroutes packets to backup next hops based on the AS-path of the BGP best path. Hence the question arises, whether in case of a fast reroute all packets that might be affected should be rerouted using Swift or to put it differently which flow rules should have higher priority FR rules or outbound policy rules.
If outbound policies have a higher priority than FR 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. Packets that do not match the policy will be redirected according to Swift
If FR rules have a higher priority, the participants outbound policies will be ignored in case of a fast reroute.
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 FR rules have a higher priority than outbound policies. Future studies 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 in the iSDX without significantly changing either the iSDX or Swift.
The convergence time upon remote failure of the iSDX was significantly reduced without requiring a lot of additional flow rules. Swift adds a small overhead to the processing of a BGP update.
Their similar architecture makes integrating one into the other intuitive 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 time of writing, only 1.8% of all IXPs have more than 256 participants. [2]
If the iSDX with Swift 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 a failed link not being encoded. One might look into implementing a more lightweight fast reroute framework that does not need to encode information on the destination MAC address.
Future work may include finding a optimal partitioning of the VMAC, enabling the participant controller to use more backup next hops and solving the conflict between outbound policies and FR rules

Bibliography

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

[2] internet exchange directory. Accessed: 2017-01-06.

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

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

[5] 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.

[6] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever. An industrial-scale software defined internet exchange point. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 1–14, 2016.

[7] T. Holterbach, S. Vissicchio, A. Dainotti, and L. Vanbever. SWIFT predictive fast reroute upon remote bgp disruptions. unpublished thesis.