ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Adrian Gämperli

Evaluating the Effect of SDN Centralization on Internet Routing Convergence

Master Thesis MA-2013-19
October 2013 to April 2014

Tutor: Vasileios Kotronis
Co-Tutor: Prof. Xenofontas Dimitropoulos
Supervisor: Prof. Bernhard Plattner

Abstract

The state of the art inter-domain routing protocol is BGP. For many applications in the Internet reliable connectivity is crucial. However, it is known that BGP has slow convergence which can result in packet loss. A new approach proposed by researchers is to form AS clusters with multiple Autonomous Systems using Software Defined Networking.
We developed an inter-domain emulation framework based on Mininet which supports SDN switches and BGP routers. Therefore it is possible to conduct hybrid SDN and BGP inter-domain routing experiments. The BGP routers run Quagga.
Moreover, we have designed and implemented a SDN inter-domain controller based on POX which not only controls the SDN switches, but also interacts with BGP routers outside the cluster. The goal of the controller was to improve the routing convergence time in the network. We run route announcements, withdrawal and fail-over experiments on the clique topology with different number of nodes, controller parameter value and percentage of SDN deployment.
We found that the AS cluster approach never had a significantly worse average convergence time than a pure BGP deployment. But on the other hand the benefits can be significant compared to the pure BGP deployment. When using our controller parameter recommendation the centralization had the biggest impact on the withdrawal experiment where we had up to 85% less convergence time. In the fail-over experiment even a small SDN deployment is beneficial. Announcements have an improved convergence time in bigger SDN deployments.

Contents

List of Figures

Chapter 1

Introduction

Current inter-domain routing is performed using the Border Gateway Protocol (BGP). This protocol is a distributed implementation of route selection, therefore every BGP routers computes the path itself. The Internet is very popular and many Internet applications depend on a reliable connection over the Internet. However, it is well-known that BGP has some problems.

1.1 Motivation

One problem BGP is facing is the slow convergence time. After a topology change in BGP it may take a long time until the network converges as routers try to find a consensus on new paths [1]. Even though there are approaches which try to modify BGP in order to improve the convergence time, is not known that anyone tried to actually deploy such solutions in practice [1] [2]. Therefore any new solution has to be able to inter-operate with BGP.
In recent years Software Defined Networking (SDN) has become popular for intra-domain routing. The SDN approach separates the routing and the forwarding plane. Consequently the routing decision can be centralized which offers new possibilities. It has been shown, that in intra-domain routing a centralized approach can improve the convergence compared to link-state protocols [3].
While SDN in intra-domain routing has been discussed, the effects of SDN in inter-domain routing are still unexplored. Kotronis et al [2] propose a new inter-domain routing approach by combining several Autonomous Systems into clusters and introducing a centralized routing decision for those participating in the cluster. By the introduction of centralization they suggest that it helps in enhancements in inter-domain routing since the routing decisions can be made with a bird's eye view. Furthermore, this approach is also backwards compatible and therefore the clusters can be gradually built and integrated within the current Internet topology.

1.2 The Task

The tasks of this project can be split into the following two subtasks:

Emulation Framework The first subtask of the project has the goal of building an inter-domain emulation framework. The framework should not only support legacy inter-domain network devices, but also SDN switches. It should feature the possibility to conduct hybrid inter-domain routing experiments.

Effects of centralization The second subtask of the project is to build an inter-domain routing SDN controller which interacts with BGP routers and is able to form clusters out of SDN switches representing ASes. Using this controller and the emulation framework developed in the first subtask, experiments should be conducted to measure the effect of centralization on the convergence time in an experimental manner.

1.3 Contributions

We make the following contributions:

Emulation Framework We developed a network emulation framework based on Mininet which not only supports SDN networks but also legacy BGP networks. BGP routers support the Gao-Rexford guideline policies [4]. The framework automatically manages IP addresses and configuration of the devices. Furthermore the user can create more realistic topologies by compiling them from public measurement data. To analyse the gathered information analysis tools are provided.

IDR SDN Controller We propose an Inter-Domain Routing (IDR) SDN controller, which is able to control clusters formed out of several Autonomous Systems. Furthermore the controller is able to interact with BGP routers outside the cluster.

Inter-domain routing convergence time The experiments conducted in the clique topology show that AS clusters can improve the convergence time significantly in certain scenarios. Already small SDN deployments can have a beneficial effect on the overall convergence time. With the recommended controller parameter we never observed a significantly higher average convergence time for a network with a SDN cluster compared to a pure BGP deployment.

1.4 Acknowledgements

I would like to thank Prof. Bernhard Plattner for the opportunity to do the master project at the Communication Systems Group. I'm especially thankful for the invaluable feedback, discussions and time invested by my tutors Vassilis and Fontas during the whole project.

1.5 Overview

In Chapter 2 we introduce the main technologies used in our project. Chapter 3 describes how we designed our SDN controller. In the subsequent Chapter 4 we give details about the implementation of the developed emulation framework. Chapter 5 describes the experiments done. Furthermore it presents the results and discusses them. Possible future work is presented in Chapter 6. We then outline related work in Chapter 7. Finally, we summarize our project in Chapter 8.

Chapter 2

Background

We first introduce the current state of inter-domain routing and explain how BGP works. In the following section we give an introduction into SDN and finally we give some information about Mininet, a SDN emulation framework.

2.1 BGP

The Border Gateway Protocol (BGP) is the de facto standard of inter-domain routing. It is a distributed inter-domain routing protocol which is defined in a couple of IETF RFCs. The most recent version is RFC 4271 [5]. In this report BGP always means eBGP. In this section we want to summarize parts of BGP which are used or are of relevance in this project.

2.1.1 Basic principles

BGP is a distributed policy-based path vector protocol. Therefore route announcements always contain the full AS-path to the destination, which a packet will theoretically take to the destination. A routing domain in BGP is called Autonomous System (AS). Each AS has a unique number: the AS number. However, an AS can operate multiple BGP routers. When a router is connected to routers of a different AS, route exchange is facilitated through a BGP 'peering'.

Update message A BGP update message can either include an announcement or withdrawal. These messages most importantly include a prefix, an action (announcement, withdrawal) and some attributes such as the AS path or the next hop. The AS path contains most importantly the AS sequence. The AS sequence contains an ordered list of AS numbers to where the packet will probably flow when it is being sent on this path.

Update process After BGP has established the connection with a peer the peers can start sending BGP Update messages to each other. Standard BGP only selects one best path, therefore no multi-path is supported. This implies that when a router is announcing a path to a destination with sequence A and afterwards sends another announcement to the same neighbor with a different sequence B, sequence A is then regarded as invalid.

2.1.2 Path selection algorithm

In BGP every router decides itself which route to choose. The choice of the best path is based on local configuration according to the AS policy, AS path length, path origin, age of the path, router ID of the neighbor, IP address of the neighbor, etc. [6]

Minimum Route Advertisement Interval (MRAI) timer The MRAI timer delays BGP updates to the neighbors in order to mitigate the problem of route flapping and achieve more stable routing. The interval is per destination and applied per peer. (see Section 9.2.1.1 in [5])

Policy Using parameters it is possible to set policies. One might want to take advantage of such possibilities due to business agreements or technical reasons such as congestion.

Loops In BGP loops can be detected as the update messages also contain the path the packet probably will take. Therefore every router can check whether the path loops back to itself by checking whether its AS number is contained in the AS path. Such update messages can then be discarded. However, information inconsistencies between the routers can lead to loops. [7].

Quagga Quagga [8] is a routing software not only implementing BGP but also RIP and OSPF. It is regarded as one of the most used routing software for inter-domain routing on general purpose hardware.
Mattia Rossi [9] describes that Quagga uses a so called 'burst' MRAI timer. This means that all updates are queued up per neighbor and are sent as soon as the timer expired. Additionally he explains that in Quagga the timers are not applied to withdrawals.

2.2 Software Defined Networking

In recent years Software Defined Networking has become popular in intra-domain routing. Because the routing decision and the actual forwarding of the packets are two different problems, SDN splits the control and the data plane. As the routing decision can be done in a centralized fashion it is possible to make more advanced decisions and routing management problems can be solved easier, while new routing alternatives and applications (e.g. 'green' routing, time-of-day routing) can be implemented.

OpenFlow OpenFlow [10] is an implementation of a southbound interface for SDN. OpenFlow consists of OpenFlow switches and controllers. Whereas the switches forward the packets, the controllers are responsible for creating decision rules and installing them on the switches. The communication between the controller and switch is done over a secure channel. The rules sent are called flow entries and are stored in flow tables on the switch. A flow entry consists of matching fields, counters and instructions. When a packet matches the matching fields the counters are updated accordingly and the instructions are applied to the packet. Matching fields can e.g. include source or destination IP addresses, as well as VLAN ids, TCP ports, etc. An example instruction is to alter the destination IP address of the packet. Figure 2.1 shows an example network with four hosts, connected to an OpenFlow switch. The OpenFlow switch is connected to an OpenFlow controller.

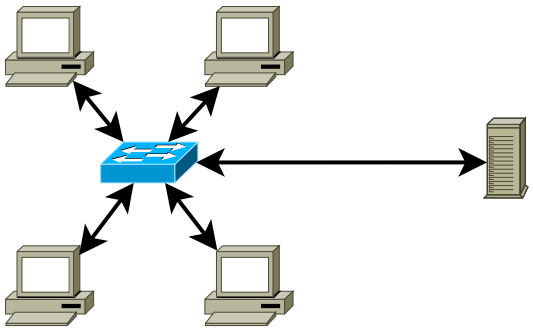


Figure 2.1: Example OpenFlow network

2.3 Mininet

Mininet [11] is a SDN emulation framework. It runs hosts as processes of the host system, which has a positive effect on the used resources. Mininet is mainly written in Python and provides a

Python API for network creation and monitoring. It can emulate OpenFlow switches, OpenFlow controllers, hosts and links along with their properties such as maximum host CPU, link latency and bandwidth, etc.

Chapter 3

Designing a SDN Inter-domain Routing Controller

In this chapter we give insight in the design of the SDN inter-domain routing (IDR) controller.

3.1 Design goals

We want to build a controller which exploits the centralization of the control plane routing decision process and build AS clusters [2]. Using this approach we want to primarily achieve better routing convergence. It is also crucial that the controller is not only optimized for convergence but also inter-operates with the currently deployed BGP routers.
The goals we want to achieve in particular are the following:

Exploit centralization As suggested in [2] we want to form AS clusters which use a centralized controller. Due to the centralization we want to primarily improve the convergence time in general, helping both the cluster and the outside (legacy) world in providing more stable routing.

Inter-operating with BGP As BGP routers are currently deployed across the globe and are managed by a lot of different parties, it is crucial that a controller inter-operates with BGP routers and that compatibility with the BGP standard is supported.

Shortest path The routing algorithm should choose the shortest path based on the number of Autonomous Systems. Our main baseline routing 'policy' is: 'prefer the shortest AS-level path' (in terms of hops).

Disjoint clusters The controller should support disjoint clusters and paths which leave and reenter the cluster. Allowing this kind of paths makes it possible that when the cluster is internally partitioned due to an inter-domain link failure, it is not partitioned on the global level since paths that join the two parts over legacy outside ASes can still be used.

No implicit policies No policy implications should be made, such as preferring internal paths over paths which are partially outside the cluster. As mentioned before we follow the simple policy of preferring the shortest AS-level path in all cases.

No cluster lock-in The AS number of the participating Autonomous Systems should be kept to avoid cluster lock-in and to provide the possibility for a smooth transition. We also keep the AS numbers of each participating AS as each AS might also want to use a different upstream provider which does not belong to the cluster as backup or for other purposes. In general, an AS maintains its 'identity' whether it is outside or inside the cluster.

No multi-path As a first step we do not support multi-path.

3.2 Path selection

To understand the operation of the path selection algorithm, we first introduce two graphs, which are important for the path selection.

Switch graph The Switch graph is a directed graph, which supports simple edges between the same source and target node with the same direction (no multi-edges). The same graph is used for all prefixes in the network. It basically represents the physical topology, seen from the controller's perspective.
We have two kinds of nodes in this graph. Firstly we have switch nodes, which represent SDN switches. Secondly we have prefix nodes. An edge shows that data can be forwarded from the source to the destination of the edge. The Switch graph is built gradually. We add a directed edge between two switch nodes, when a switch node detects a link in that direction. An edge from a switch to a prefix node is added, when the prefix is learned from BGP or the prefix is directly connected to that particular SDN switch (origin of prefix). As there can only be one edge per direction between two nodes we only add the best path. On edges learned from BGP we also store the AS sequence as an edge attribute.
Figure 3.1 shows the process of selecting the best path between a switch and prefix node. It also shows what is altered in the Switch graph. Generally we save all paths which the clusters receives information about. When a removed path has been the last path to that prefix, the path is removed from the Switch graph. However, when it is not the last path to reach the same destination we update the switch graph with the best saved path. All newly discovered paths are saved. When the new path is a BGP update we always update the Switch graph with the best path, even if the best path has not changed. This is due to an implementation specific issue. Directly connected paths are only triggering a Switch graph update when the new path is better than all saved paths.

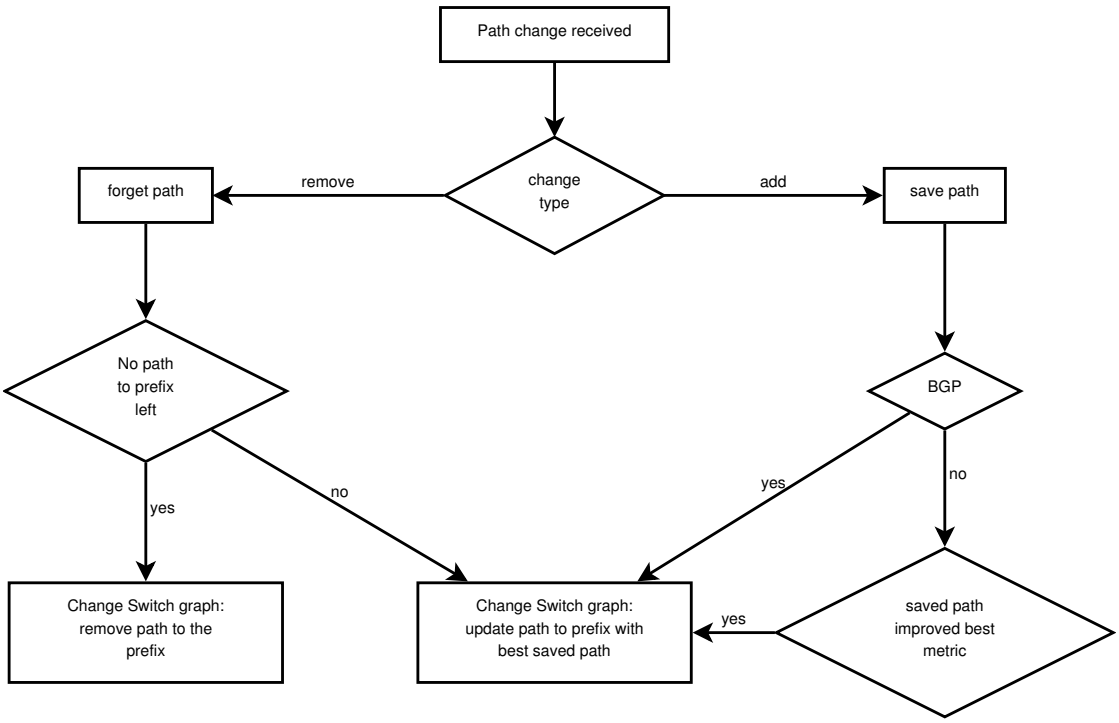
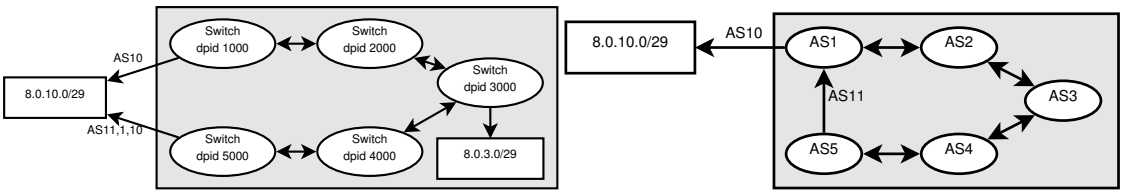


Figure 3.1: Switch graph updates

Figure 3.2a shows an example of a Switch graph. Switch 1000 to 5000 are connected in a sequence and form a cluster. Both switches 1000 and 5000 know a path to the prefix 8.0.10.0/29, which they learned from BGP. Switch 3000 has a directly connected prefix (8.0.3.0/29).



(a) Switch graph (b) AS topology graph: 8.0.10.0/29

Figure 3.2: Example of redrawing an edge

AS topology graph The AS topology graph saves the AS topology within the cluster for a specific destination prefix. Therefore there can be different graphs for different prefixes. This graph is a transformation of the Switch graph. Like the Switch graph, it is a directed graph.
This graph consists of cluster ASes and a destination prefix node. At the beginning of the transformation to the AS topology graph all AS numbers of the cluster are added as nodes. The AS relationships inside the cluster, which have been represented as edges between switch nodes in the Switch Graph are also added to the new graph. Then all edges between switch nodes and the destination prefix node in the Switch graph are checked regarding whether the AS sequence contains AS numbers of the cluster. When no cluster AS number is found we add a link between the AS node of the switch and the prefix with the number of BGP AS hops as weight. However, when such an AS number is found, a new edge is added between the AS node of the previous source of the edge and the first AS number which has been found to be part of our cluster. The edge weight is the number of hops plus one, since we would have one edge more between those AS numbers. We also store the modified AS sequence as an edge attribute, and the remaining AS sequence is discarded. In Figure 3.2 an example transformation from a Switch graph to a AS topology graph for the destination 8.0.10.0/29 is shown.

Algorithm To compute the best paths Dijkstra is run on the AS topology graph using the previously mentioned weights. As Dijkstra never creates loops we will not create paths with loops within our cluster.
We introduced the two previously explained graphs for loop avoidance regarding paths that pass through our cluster, exit it and revisit it (such paths can be valid or not depending on whether they are loop-free or not). BGP discards paths which include the router's AS number. However, this is not possible in our cluster environment, since Dijkstra needs to run on the same graph for every node, to not introduce unintended routing behavior. When discarding all paths with AS numbers of the cluster in the AS sequence we would not support disjoint clusters. Therefore both graphs are necessary.

Recomputation Wait Interval We added a delayed recomputation of paths instead of delayed BGP updates to always do what we are announcing.
We added this mechanism for the following reasons: Firstly we want to avoid loops with neighbors due to outdated information. Secondly because it seems likely that in a standard topology our cluster receives another announcement about the same prefix through other neighbors. Therefore we can make the network more stable. Furthermore we can reduce the number of path changes.
Before we queue a recomputation, the prefix to be recomputed is checked whether it is already contained in the queue. When this is the case it is discarded otherwise it is added to the queue. Queued prefixes are recomputed after a fixed recomputation wait interval (RWI). Figure 3.3 shows this process.

Path recomputation Paths are only recomputed when needed. Figure 3.4 shows which type of Switch graph change, result in which prefixes to be requested for recomputation. A link change between switch nodes or a switch change in the Switch graph request a full recomputation of all prefixes currently known in the network. However, when a prefix is changed, e.g. a second switch adds a path to a prefix, only that single prefix is requested for recomputation.

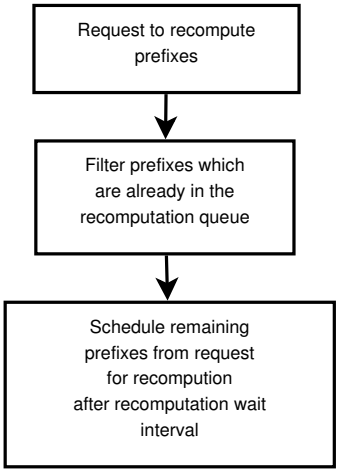


Figure 3.3: Recomputation Queue

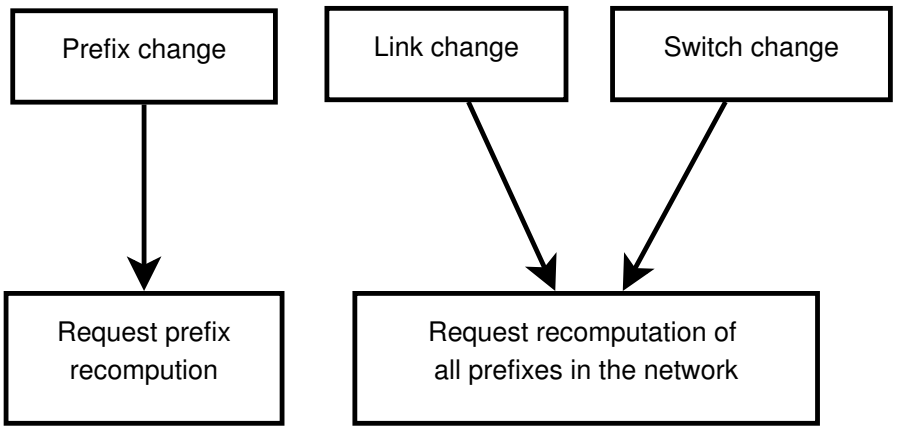


Figure 3.4: Switch graph changes

Consistency To achieve consistency during route installation we install the routes from the destination to the source. However, we do not wait until the SDN switch confirms (OpenFlow barrier) the installation as we faced timing issues. Without this ordering we might get a possible policy violation, unintended paths or loops [12].

Inter-operation with BGP We announce the prefixes at the same time to the BGP neighbors as we do the flow entry installations on the cluster SDN switches without a delay. Announcements contain the complete AS sequence which also includes the AS numbers in our cluster as BGP would do. Therefore a path change within our cluster with the same length is also announced to the neighbors. To minimize the number of flow installations and BGP announcements we check whether we already sent it before. Figure 3.5 shows how a flow entry change is processed within the controller as described in this paragraph.

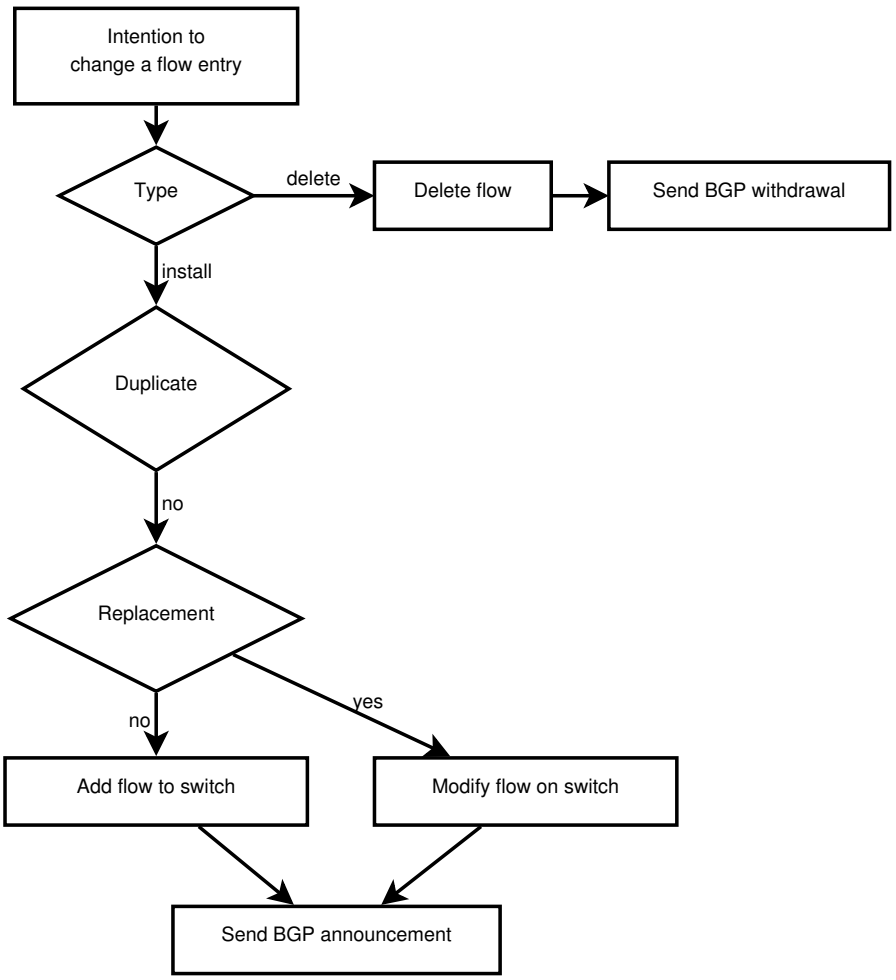


Figure 3.5: Flow entry change

Chapter 4

Hybrid SDN&BGP Emulation Framework

We developed a hybrid SDN and BGP Emulation Framework. Our framework is based on Mininet [11] and POX [13]. POX is an OpenFlow controller. The framework is written in Python and runs on Ubuntu 13.10 x64.

4.1 Overview

In Figure 4.1 we show an overview of our implementation. We see an example of how the hosts are connected to each other. On the left side we see the legacy part of the test network. Whereas on the right side we illustrate a sample cluster. Both, BGP routers and SDN switches, can originate prefixes. It is also possible to add a host which is connected to the network device and has an IP address within the particular prefix. Furthermore both approaches can be combined to form a hybrid network. In Figure 4.1 for simplicity we omit the event collector which collects events from all parts of the network and saves them into a log file. BGP routers always peer with a BGP monitor, which has the goal to collect routing control plane state changes. Moreover, per cluster we have a special cluster BGP router which relays routing information between BGP routers and the SDN controller. For every BGP peering there is a link from the cluster BGP router to the adjacent SDN switch. In our network design we model one Autonomous System (AS) as one router. Therefore the effects of intra-domain routing are neglected. The reason for choosing this is that we want to isolate the effect of inter-domain routing convergence and experimentally study its properties.

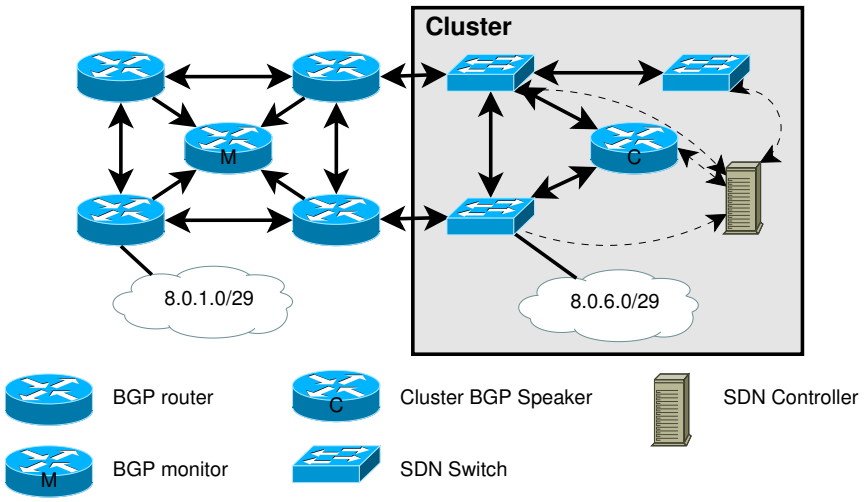


Figure 4.1: Emulation overview

4.2 Controller

In this section we give insights in the implementation of the controller described in Chapter 3.

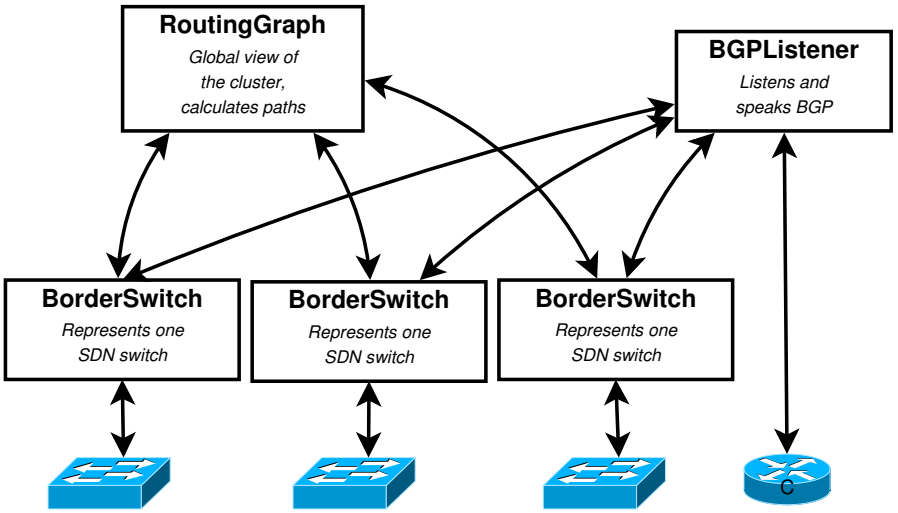


Figure 4.2: Simplified conceptual controller implementation overview

Overview In Figure 4.2 an overview of the implementation is shown, which is slightly modified for better understanding. Every OpenFlow switch which is connected to our controller has a instance of the class BorderSwitch to keep state of information related to that switch. These instances also exist for SDN switches which do not have a peering to a BGP router. One important task of the instance is to keep track of prefixes it has learned either because the prefix is directly connected to that switch or because it has learned that network from BGP. Best path changes are then propagated to the RoutingGraph instance, which keeps the global overview of the network. The Switch graph is stored in that instance and is updated with the received best path changes. As soon as flow entries have been calculated the RoutingGraph passes them to the BorderSwitch instances which handle them. They not only install the flow entries on the OpenFlow Switch but also announce and withdraw paths over BGP to its neighbors. This includes keeping track of the announced paths, which may be sent to a newly established BGP peering neighbor. Furthermore the BorderSwitch instances respond to ARP requests of hosts.

Forwarding flow entries Flow entries responsible for routing the payload traffic are installed permanently on the switches. We set the network destination and the Ethernet frame type (IPv4) as match fields. Furthermore we set the priority to the prefix length to prefer more specific prefixes over larger prefixes. The flow entries only have an output action to forward the traffic to the destination. When the next hop is a BGP router or a directly connected host also the destination MAC address is changed.

Graph manipulation To store the graphs the DiGraph class of NetworkX is used. From the same library the Dijkstra implementation is used for shortest path computation.

BGP communication Every cluster has one BGP host which is responsible for the BGP communication with all BGP neighbors of the cluster. This is however not a regular BGP router as we need to parse all incoming paths and inject announcements. We therefore use ExaBGP [14], which does not install the forwarding rules on the host but passes the paths to an application. Because we only have one ExaBGP host per cluster and a static peering prefix length, the host has one link per BGP peering to the adjacent switch. The switches detect the BGP connections based on the peering IP addresses which are automatically provided to the controller. Therefore the controller can detect and learn the switch port numbers and start forwarding ARP and IP traffic between the external BGP router and ExaBGP of the cluster. These rules are installed to enable a successful BGP connection.

Cluster-wide topology detection The POX OpenFlow discovery module is used for link detection between switches. As we had some problems regarding flapping connections and the topology within the cluster is not changed intentionally in our setup, we changed the timeout setting from the OpenFlow KeepAlive module to Python's maximum integer. In case it is still flapping, we log those events to the log file and they are detected as an error while analyzing for convergence time.

Origination of a prefix As it is unknown to the controller on which switch port a host with a prefix is connected to, the controller waits until it receives a packet belonging to the prefix and then starts announcing the prefix.

Limitations The controller does not yet support policies and only announces prefixes when a host has been added to that network. Furthermore it does also not fully support AS sets because they are not used in our network. POX is implemented using cooperative multitasking. Therefore the controller does no parallel task execution.

4.3 Emulator

The goal of our network emulation was to have an easy to use emulator which can be easily adapted.

4.3.1 Mininet extensions

For our emulation framework we use Mininet (see Section 2.3). Mininet has been chosen because of its performance advantages and it provides an easy to use Python API to generate SDN networks. As we also wanted to emulate BGP networks we extended Mininet to also use BGP routers using the Quagga routing software. Furthermore some code changes to Mininet were needed.

Naming

Host names All our networking devices, which represent an AS (BGP routers and SDN switches) are named 'r AS number . Router ID '. The Router ID is an incrementing identifier within the AS. In the current framework the Router ID is always equal to 0, as we only have one network device per AS. With this naming pattern we allow further extension of our framework to support multiple devices per AS. We have chosen the same naming pattern for both, BGP routers and SDN switches, because this allows experiment scripts to refer to a particular AS network device without requiring to know which type it will be. The dpids of the SDN switches are numbers and can be calculated using AS number $* 1000 +$ Router ID.
Hosts are named in a similar way as network devices: 'h AS number . Host ID '. The Host ID is also unique within the AS. The naming has also been chosen because of easier and faster management and pattern recognition.
And as it is common in Mininet the controllers are named with 'c Cluster ID ' and the cluster BGP router is called 'vb Cluster ID '. The Cluster ID is a unique number which identifies a cluster. Therefore e.g. c1 and vb1 are in the same cluster.
The BGP monitor has the name 'm Monitor ID '. However, only one BGP monitor is started, therefore it is m1.

IP addresses All IP addresses are grouped by their prefix for better understanding. Moreover, we only support IPv4 addresses.

1.0.0.0/8 As BGP routers need a router id assigned we selected this range for BGP router ids.

2.0.0.0/8 Every BGP peering network is assigned a /30 prefix from this range.

8.0.0.0/8 All routed prefixes in our emulation are in this range and have a prefix length of /29. In our emulation we do not have a minimum prefix length restriction of /24 for a successful announcement or any other pre-installed policies. The second and third octets of the prefix represent the AS number. Because we use 2 octets for the AS representation we have a restriction on AS numbers to 16 bits. This is why there is no support for ASN-32.

Auxiliary

Our framework uses the the directory /tmp/sc to create configuration files, mount directories and communication sockets. UNIX domain sockets are used, because all hosts share some parts of the file system. We can not use the local interface as hosts have their own network namespace.

Mininet Command line

We extended the Mininet Command line to support some more functionality:

comment Sends a comment event to the log file, can be used to document an experiment.

marker Sets a marker in the log file, which can e.g., be used to measure the time between two markers.

bgpannounce Announces a prefix on a BGP router

lastrcps Shows number of seconds since the last routing control plane signaling change which is basically the last BGP update seen in the network. This command can also block until a specified amount of seconds have elapsed.

lastrcpsc Similar as lastrcps, but it uses the last routing control plane state change which is a SDN flow entry change or a BGP update to the BGP monitor.

waitconverged Waits until the network is converged. We defined converged as that 90 seconds have elapsed since the last routing control plane signaling and routing control state change. 90 seconds were selected as a practical waiting time for convergence verification since we noticed that a BGP update after 90 seconds correlated with an old routing change is extremely unlikely to happen.

link Brings both interfaces which connect two nodes up or down, depending on the experimenter's requirements.

ip Prints the IP address of a node.

wait The console blocks for a given number of seconds.

nodecmd Runs a command on a node.

interface Prints interface names which connect two nodes.

hping Ping a host from another host.

neighbors Show neighbors of a node.

checkHosts Checks connectivity between all hosts using pings.

pingHosts Ping every host from every host.

Most commands support host name substitution.

Randomness

On startup of the network several parameters are pseudo random to try to make the BGP decisions more independent from our emulation framework implementation. We shuffle the host, switches and link list of Mininet (to change the startup order), choose a pseudo random BGP router id and randomize BGP peering IP addresses. The seed is stored in the log file and can be passed to the startup script.

Wait converged

When the emulation starts we wait until the network has converged. We use the previously mentioned *waitconverged* command, which waits until the last message is older than 90 seconds. Sometimes it is necessary to add a wait command before the *waitconverged* command as it can take a couple of seconds until the network reacts.

Failure detection

To detect failures in our framework we added ping checks between all hosts. This failure detection is executed at the start and end of the experiment. When the ping check before the experiment is unsuccessful, the experiment is immediately halted for further investigation. To use this feature the user has to add a host to every AS. Furthermore we try to catch all exceptions and log them to the log file. To avoid unnoticed failures the analysis tool fails when it detects an error or certain events in the log file, unless this setting is overwritten.

BGP router

The BGP routers run Quagga. The configuration is generated automatically. We developed a Quagga template configuration which allows to easily set policies. Three policies are defined: Peer (announce own and customer's prefixes), Customer (send all prefixes), Upstream (announce own and customer's prefixes).
To allow asymmetric routing the *rp_filter* has been disabled on all interfaces. Furthermore the connect timer has been set to 5 seconds for faster startup. The BGP MRAI is unchanged at the default value of 30 seconds because in reality most routers use that value (i.e., as best practice). To have an insight in what is going on in the network and to determine the last routing control plane signaling change we capture BGP packets on all interfaces of the BGP routers and log them by sending the contained information to the event collector.

SDN switches

We use the standard Mininet SDN switches which make use of Open vSwitch [15].

4.3.2 Network setups

To save the experiment context in a consistent and reproducible way we save all network topologies and experiments as Python code, which is located in the folder nwsetup. In those files the topology and the experiments can be defined. To customize the topology command line options can be provided at every emulation start. Experiments can be defined with Python code, since this is very helpful for automation. In experiments it is possible to use the Mininet CLI commands and commands for loss measurement.

4.3.3 Topologies

There are two different possibilities to create the topologies. Firstly they can be set manually and secondly the can be created based on measured data of the Internet.
When manually creating the topology the network devices and links can be defined. Additionally it is possible to add attributes like latency to the links or annotate them with relationships (e.g., p2p, c2p) and form clusters.
The CAIDA AS Relationship [16] and the iPlane Inter-PoP link dataset [17] are used to construct the topology based on Internet measurements. The dataset from CAIDA is used to determine the links and relationships. Whereas the iPlane dataset is used to determine the latencies on the links between the ASs. On links which have no latency specified or an invalid latency has been set (such as <0ms), the default latency is used. Generally we use a default latency of 24ms, which is the the average latency of the file of 14th December 2013. The value is rounded up from 23.6459ms to 24ms. The AS relationships are used to choose the routing policy. In artificial topologies, as the ones used in the final evaluation, default policy is to prefer the shortest path route (in terms of AS hops).

4.3.4 Usage

To use the emulator framework one can use the sc-network command. The first positional argument is the network setup name, which is the name of the network setup file. The other arguments are optional. They include the configuration for the network setup or the experiment which should be made automatically. When no experiments are given the framework starts the modified Mininet console. Other possible arguments are to specify the seed or are log storage specific.
The usage is as follows:

```
sc-network [-h] [--config CONFIG] [--group GROUP] [--run RUN]
           [--experiments EXPERIMENTS] [--log LOG] [--seed SEED]
           setup
```

4.3.5 Measurement framework

Log files are generated automatically with information about the network. It was our goal to try to save all relevant information about the experiment. Furthermore we wanted to make it simple for automated analysis to not spend time on simple analysis tasks.

Event collector

An event collector has been developed where we can send events to. Events are Python classes. To save events to the log files they are serialized using pickle[1]. Events can also give context to other events. For example, an IP assignment event can provide the IP address to host name relationship to other events, which can use this information to print easier to understand log messages for humans.

Log storage structure

For better log file management the log files are sorted into directories. The format of the path is as following:

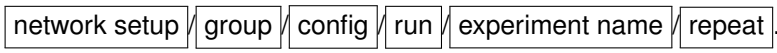


Figure 4.3: Log storage path struture

The network setup is the name of the network setup (see 4.3.2). The group can be used to group some executions of the experiments. This can be used to get a better view. Config is the configuration string which has been provided to setup the network setup. A run is a complete restart of the experiment including the start-up phase. The log entries of the start-up phase and manual experiments are saved in the run folder. However, automated experiments are saved in a separate folder. The file name corresponds to the repeat number. The difference between a run and a repeat is that a run has its own start up phase, whereas a repeat is without one. This could be the case when snapshotting would be implemented. Snapshotting would have the advantage that the state after the start-up phase can be used for multiple experiments.

Reading log files

As files are saved in a structured way log files and experiments can be easily selected for automated analysis matching certain criteria. File selection can be done using the FileManager class. Criteria can be specified which the log files have to fulfill, then all matching log files are returned. The log files can be passed to event readers which deserialize and analyze the log files.

[1] https://docs.python.org/2.7/library/pickle.html

Log file analysis

We developed some tools which can be used for log file analysis:

sc-convergence To calculate the convergence time of several different experiments. It draws a boxplot and prints some statistical information about the experiments.

sc-networkgraph As we also store network topology events in log files we can show a graph of the created network.

sc-logview To manually review the log file.

sc-bgpview To visualize BGP state changes.

Events

Events always save the time, when they are created and a unique identifier, which can be used to gather further information about the event.
The most important events are the following:

BGPUpdateEvent Quagga changes its selected route.

BGPUpdateCaptureEvent A captured BGP update message.

SDNUpdateEvent A switch installed a new flow entry.

MeasureResultEvent Saves the result from a loss measurement.

HostPingErrorEvent Usually indicates that something went wrong, as not all hosts responded to pings.

Many more classes are defined and are usually straightforward to understand.

BGP monitor

To collect the latest routing control state changes we setup an ExaBGP host which peers with every non-cluster BGP router. The advertisement interval of the BGP router is set to 1 second for that link, to receive most state changes. Therefore state changes more frequent than 1 second are not tracked.

Loss measurement

We developed a tool to measure loss. The measurement is done using a measurement server and a client. A server instance runs on every host. The protocol is built on UDP. After the initial setup phase it periodically sends a sequence number and the send time to the server. The server keeps track of these packets. It discards packets which arrived later than the buffer time. It is important that at the end of the measurement the packets are not lost otherwise the measurement can not be stopped. An issue is that while sending packets we are facing clock inaccuracy problems.
We can measure:

- packet loss (absolute)
- packet loss (relative)
- loss time

We define the loss time as the time difference between the first loss and the last loss in a measurement. The send time of the packets is used for this calculation. The loss in Figure 4.4 is 3.

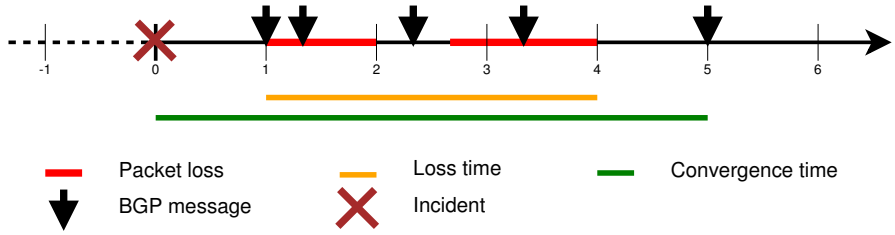


Figure 4.4: Loss and convergence time

Convergence time

The convergence time is the time between the incident and the last BGP announcement which has been triggered by an *incident*. After the convergence time the network is stable again. In Figure 4.4 the convergence time would be 5.

4.3.6 Third party modules

The controller is implemented using the libraries POX [13] (Version carp), NetworkX[2] and ExaBGP [14] (Version 3.3.2). POX is an OpenFlow controller. We use a patched version, which does not pass certain events to our control application before the connection is up and drops a BarrierIn on connection setup. NetworkX is a Python library for graphs. ExaBGP is a BGP speaker which passes the parsed BGP messages to other applications. We were able to identify some bugs and contributed patches to fix these problems. These included for example that passed BGP announcements have not been announced or have been sent to wrong BGP routers.
As already noted Mininet [11] is used with some patches. The interested reader should be aware to use a kernel version which is compatible with Mininet (we used 3.11.0-19-generic), otherwise Mininet might block after some time. Furthermore Quagga has been used, which is installed from the Ubuntu repository (Version 0.99.22.1-2). For inter process communication we use the Python binding of 0MQ also from the Ubuntu repository. Other used Python modules include: netaddr, scapy[3], twisted and numpy. For network visualization we use BGPplay.js[4].

[2] http://networkx.github.io/
[3] http://www.secdev.org/projects/scapy/
[4] http://bgplayjs.com/

Chapter 5

Evaluation of convergence time

For the evaluation of the convergence time we used the clique topology (see Figure 5.1). The clique, or a full mesh, is a network where every node is connected with every other node. We have chosen the clique as it is the worst case in terms of high BGP convergence.
The experiments were run on virtual machines with 8GB Ram and 4 CPUs (at least 2 GHz).

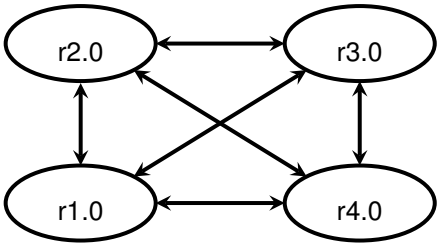


Figure 5.1: Clique topology with 4 nodes

The prefix we are investigating is connected to a BGP AS which is not part of clique but is connected to it. We will refer to that AS as AS40000. When having a hybrid clique network, the SDN nodes always have higher AS numbers than the BGP nodes. For instance when having 8 nodes in the clique and 25% SDN nodes, then AS1 to AS6 are BGP routers whereas AS7 and AS8 are in the SDN cluster. We are only using a single SDN cluster. Furthermore we installed a policy on AS40000 to only allow prefix announcements, which are originated in that AS. The links between the clique nodes have a latency of 24ms which is an average latency.

5.1 Experiments

We defined the following three experiments.

5.1.1 Withdrawal

In the withdrawal experiment the announcing AS has only one upstream. AS40000 is only connected to AS1, which is a BGP AS. The experiment is to take down the link between these two ASes. The experiment is illustrated in Figure 5.2. We selected this experiment because it is known that BGP withdrawals have the longest convergence times. [1]

5.1.2 Fail-over

The fail-over experiments tests what happens when one of the two upstream links fails. AS40000 is connected to AS1 and the highest AS number in the clique. Therefore the investigated AS is connected to both network types (BGP, SDN), if present in the network. This experiment is a typical case of an ISP which may experience a failure with the peering of one of its upstreams. Figure 5.3 shows this experiment.

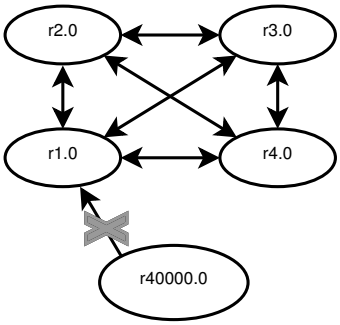


Figure 5.2: Withdrawal experiment

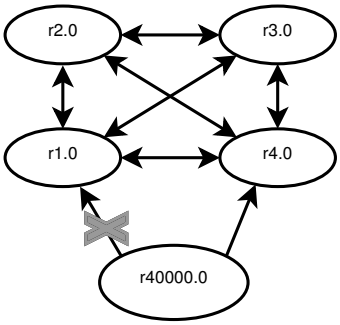


Figure 5.3: Fail-over experiment

5.1.3 Announcement

Having one upstream to the clique (see Figure 5.4) the AS starts to announce a new prefix.

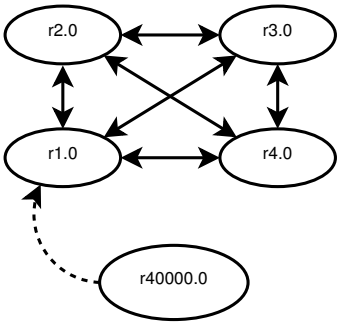


Figure 5.4: Announcement experiment

5.1.4 Changing Parameters

Number of nodes We expect different results for a different number of nodes as the number of peerings depends on the number of nodes. We have therefore chosen to do the experiments for 8 and 16 clique nodes.

Percentage of SDN deployment We vary the percentage of SDN switches in the clique as we want to see how the convergence time is influenced by changing the number of ASes that join the cluster.

Recomputation Wait Interval We also want to explore the effect of the Recomputation Wait Interval (RWI) on the convergence time of the network and find a good practice value.

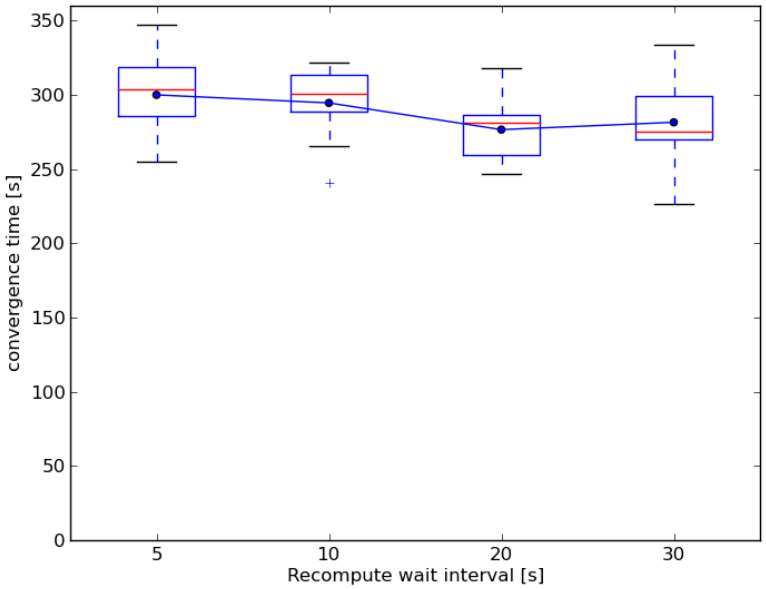


Figure 5.5: Withdrawal: 25% SDN, 16 nodes

5.2 Results

In this section we present the results of the experiments. The experiments always run ten times for statistical confidence reasons. The results are presented in boxplots. Lines connect the average values in order to exhibit the 'trends' that characterize the plots.

5.2.1 Withdrawal

Recompute Wait Interval

In Figure 5.6 and Figure 5.8 it seems that smaller RWI values are better for convergence time in cliques with 8 nodes even though the clusters participate in path exploration. However, especially interesting is comparing the convergence time with 30 seconds RWI of those both plots. With 25% SDN deployment BGP creates pathological paths within the BGP part of the network before the first RWI elapses. At the first recomputation the cluster pick up the pathological paths and reannounced them. However with the 50% SDN deployment the pathological paths include BGP numbers of the clusters at the time when the cluster starts its first recomputation. As cluster numbers are included in the AS path, the cluster can detect that these are pathological paths and sends a withdrawal to all its neighbors at once. After these withdrawals the BGP routers also start to withdraw the prefix to certain neighbors until the full network converged. In all these cases the average convergence time is about the RWI value. We also noticed the same phenomena for 30 seconds RWI in Figure 5.9. The two outliers are a multiple of the RWI as the previously mentioned phenomena only happen when the RWI was triggered the second time. The fact that in Figure 5.10 the convergence time corresponds to the RWI seems obvious since only one BGP router is participating in path exploration.
The convergence times for 16 node cliques (Figure 5.5, Figure 5.7 and Figure 5.9) have a slight trend to be better with higher RWI values. Which we think is due to the less frequent contribution to the path exploration process.

SDN Deployment

By looking at Figure 5.11, which is a 16 nodes clique, we can see the convergence time drops almost linearly with the percentage of SDN deployment. Comparing the convergence times between the pure BGP and 75% SDN deployment one can notice a huge convergence time

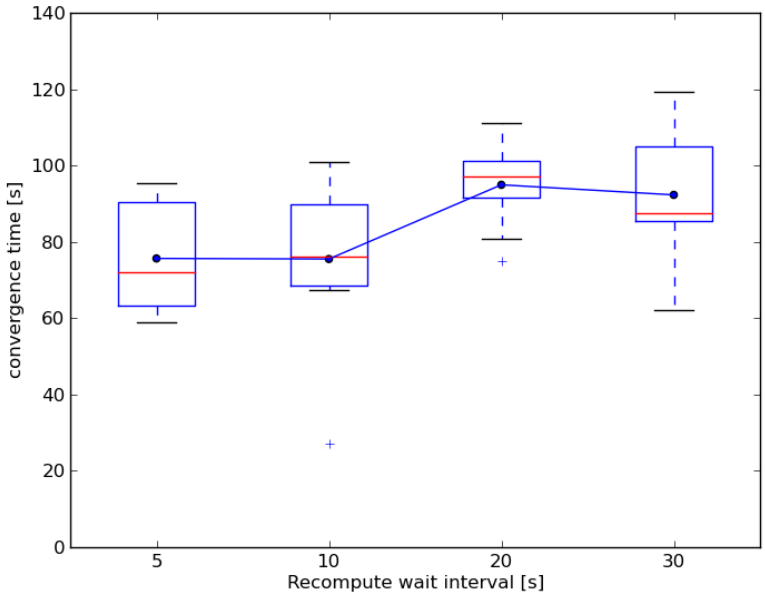


Figure 5.6: Withdrawal: 25% SDN, 8 nodes

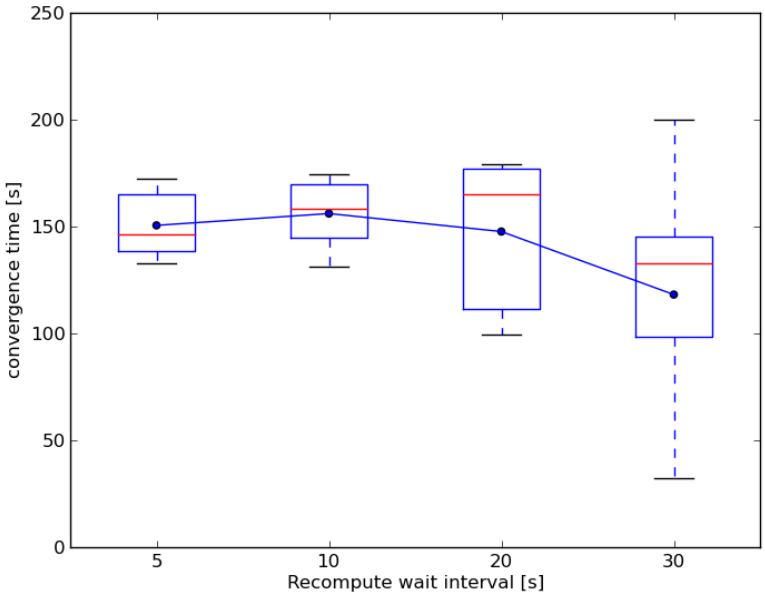


Figure 5.7: Withdrawal: 50% SDN, 16 nodes

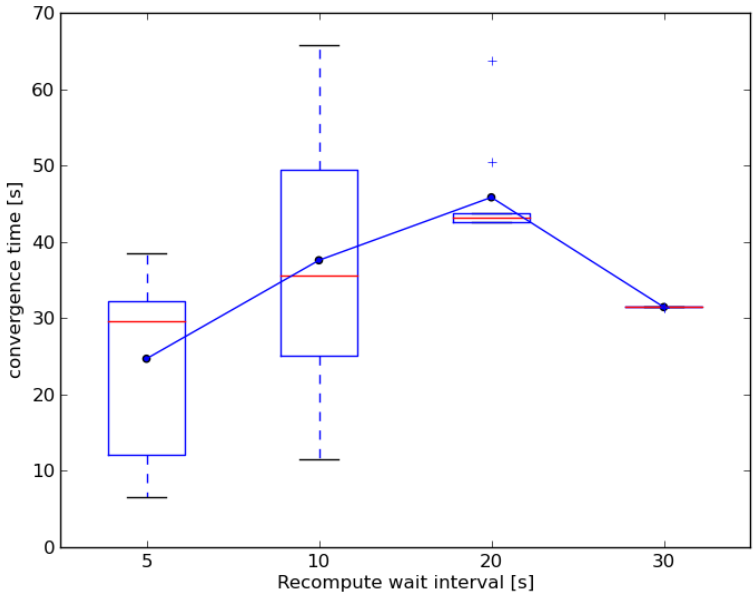


Figure 5.8: Withdrawal: 50% SDN, 8 nodes

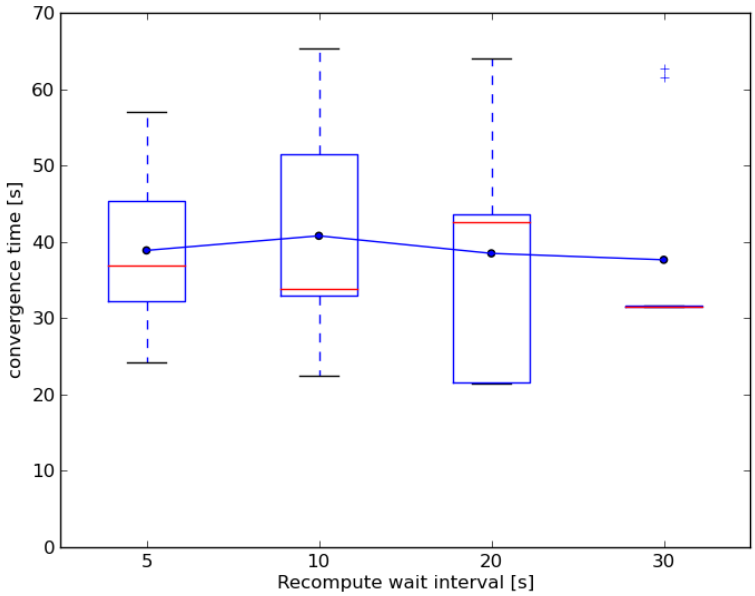


Figure 5.9: Withdrawal: 75% SDN, 16 nodes

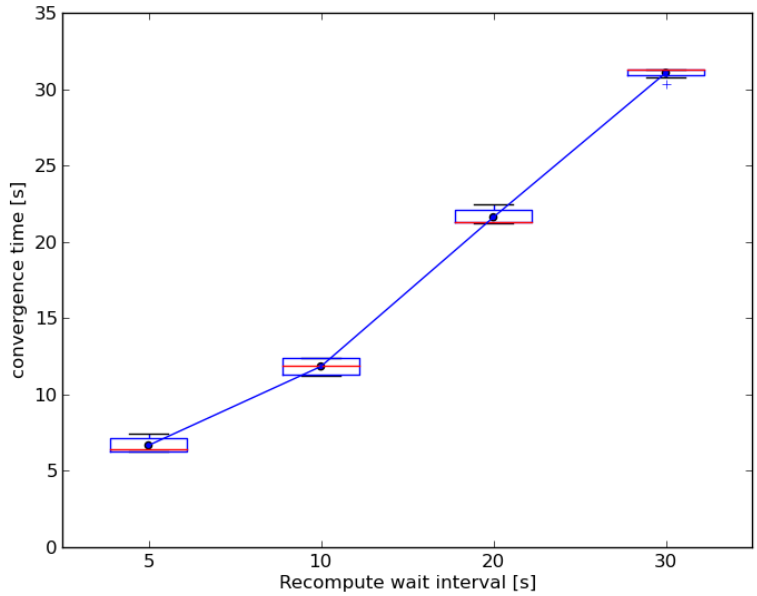


Figure 5.10: Withdrawal: 75% SDN, 8 nodes

improvement of about 5 minutes. According to Figure 5.12 a smaller network with 8 nodes is behaving similarly in terms of convergence times, except that it has shorter convergence times overall. These observations similarly apply for the other RWI values tested. Figure 5.13 shows very clearly the lower bound of the convergence time for our cluster. The convergence time can not become smaller than the RWI value as our cluster will not send a route update about a prefix to BGP neighbors before the RWI timer expired.

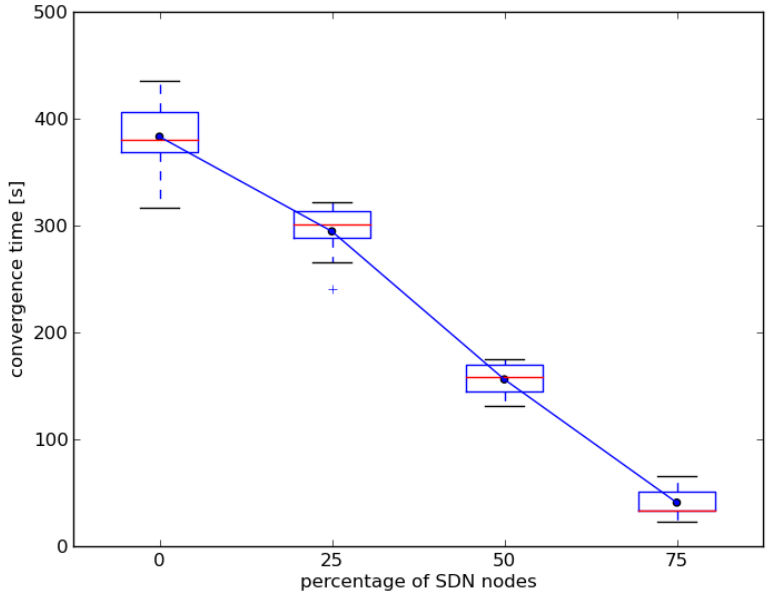


Figure 5.11: Withdrawal: 10s RWI, 16 nodes

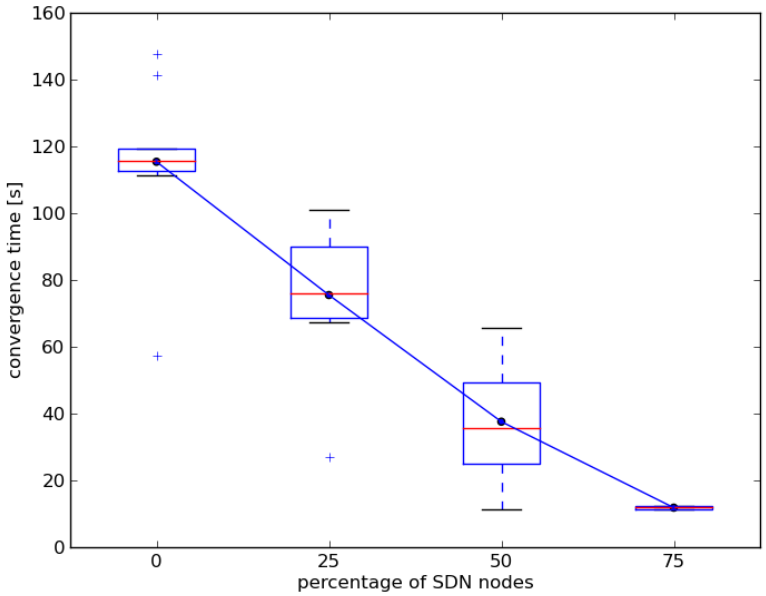


Figure 5.12: Withdrawal: 10s RWI, 8 nodes

Interpretation

We see that in the withdrawal process in networks with a small number of BGP nodes (≤ 4) a cluster can reduce the convergence time by keeping an old (shorter pathological) path so that BGP routers change their pathological paths to the cluster before the RWI expires. Due to loop detection of the controller, the controller can detect that all nodes changed the best path to the cluster and withdraws the prefix to all BGP neighbors at once when the RWI expires. Due to this, all BGP routers receive withdrawals from all SDN neighbors. They then only have paths over BGP neighbors left. From which they select the best path. Because all pathological paths they can choose from have the same length, Quagga needs a tie breaker. Separate experiments suggest that Quagga uses the neighbors router id as tie breaker. As every node is connected with every other node all BGP routers choose the same neighbor as next-hop. The nodes then send a withdrawal to the selected next-hop, as they have to withdraw the previously announced path. Announcements to other neighbors are not yet send because of the MRAI timer. The selected next-hop obviously receives withdrawals from all neighbors. It therefore can send withdrawals of its pathological paths to all neighbors. This can repeat itself until the MRAI of the BGP routers or the RWI of the cluster expires.

Looking at these results it becomes obvious that in future work we have to look into networks where not all BGP routers are directly connected to the cluster. However, it seems to be a good idea to send BGP updates for the same prefix for all cluster members at the same time.

To recommend a RWI in this case is difficult because for bigger cliques we would need a higher RWI and for smaller ones a lower RWI. However, even with worse RWI values the convergence times are not very different from the good ones. Therefore we suggest to not depend the RWI choice on this experiment. Even though the different RWI values do not change the convergence time much the convergence time decrease a lot due to introduction of the cluster. Even a small percentage of deployed SDN ASes (25%) is already beneficial for better convergence times. Furthermore the plots show the impact of the SDN controller is significant and can improve the convergence time a lot.

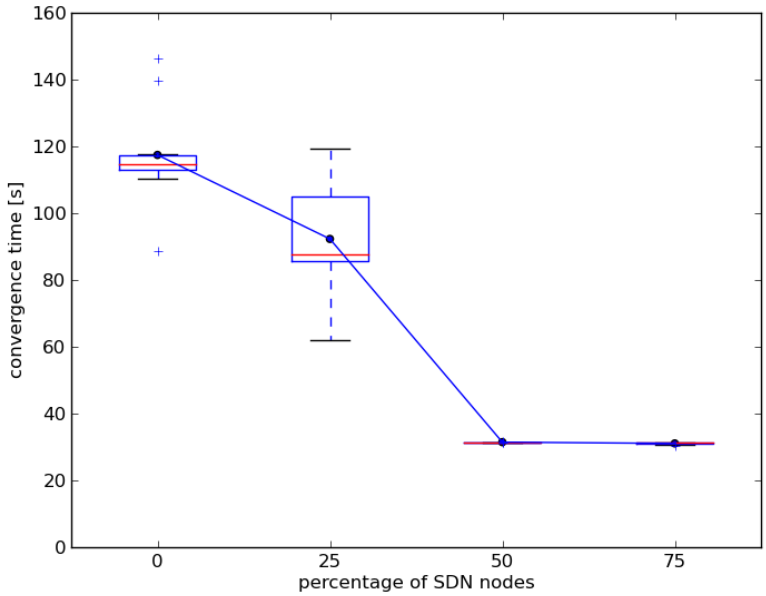


Figure 5.13: Withdrawal: 30s RWI, 8 nodes

5.2.2 Fail-over

Recompute Wait Interval

When first comparing plots where we have 8 nodes in the clique, we can see that in Figure 5.15 (25% SDN deployment) and Figure 5.17 (50% SDN deployment) we have the best average convergence time with an RWI value of 10 seconds. But when looking at Figure 5.19, where we have 75% SDN deployment, we see the best average convergence time with a 5 seconds RWI. In the 16 nodes clique experiments (Figure 5.16, Figure 5.16 and Figure 5.18) the 5 and 10 seconds RWI experiments tend to have better convergence times than in the other cases.

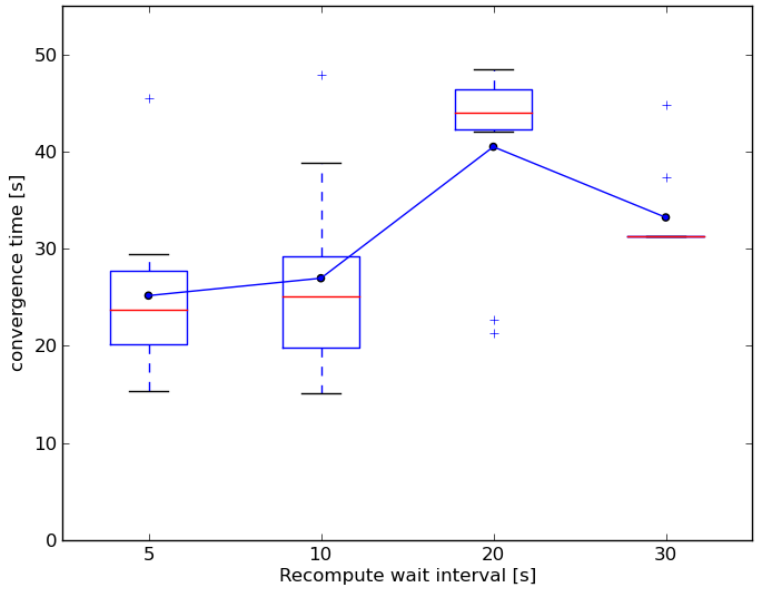


Figure 5.14: Fail-over: 25% SDN, 16 nodes

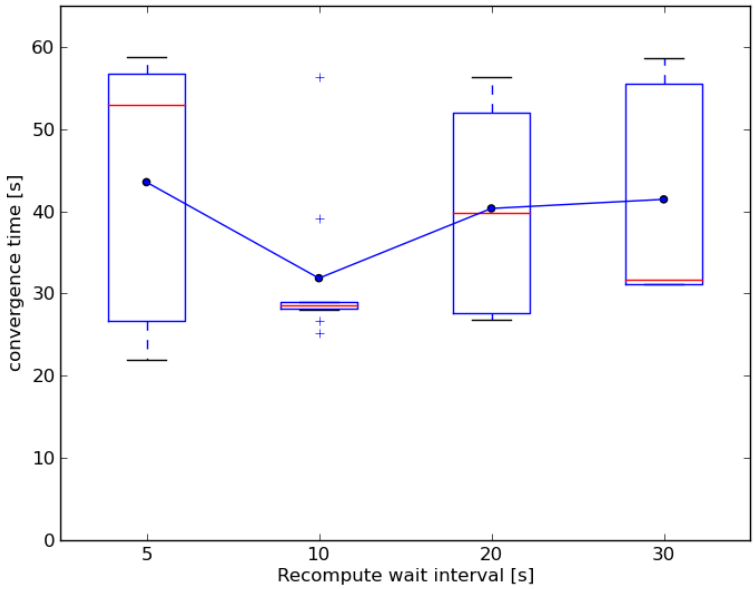


Figure 5.15: Fail-over: 25% SDN, 8 nodes

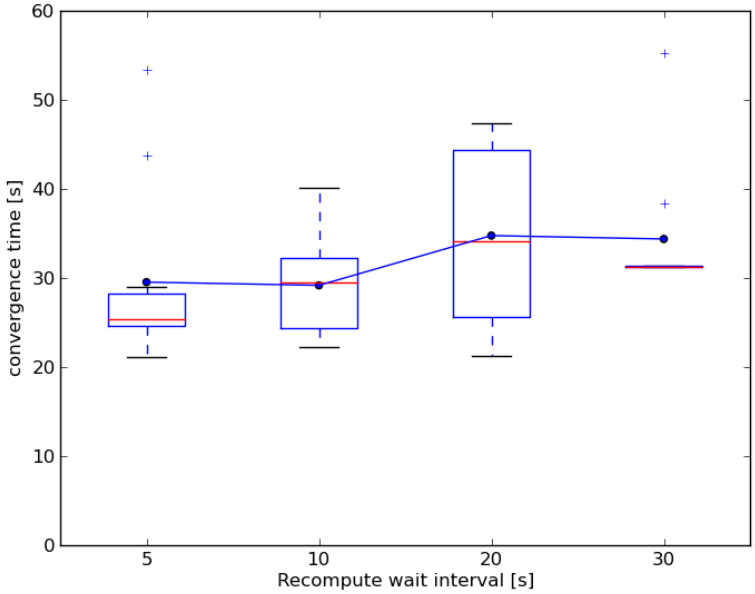


Figure 5.16: Fail-over: 50% SDN, 16 nodes

SDN Deployment

Figure 5.20 and Figure 5.21 with a 10 seconds RWI show that when we have 25% SDN it is already about the best convergence time result for that the 10 seconds RWI and it is better than the pure BGP deployment

Interpretation

We have seen that most times we have approximately the best convergence times with 10 seconds RWI and with that RWI value we would have a small positive impact with 25% SDN de-

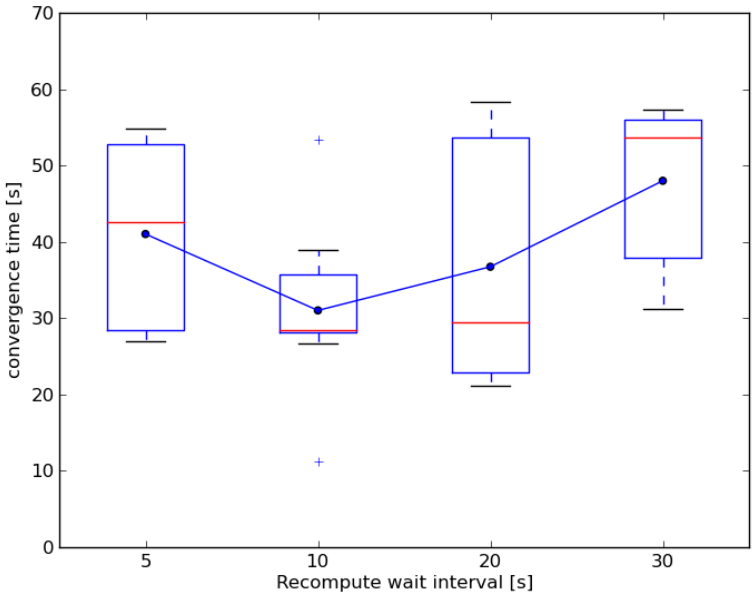


Figure 5.17: Fail-over: 50% SDN, 8 nodes

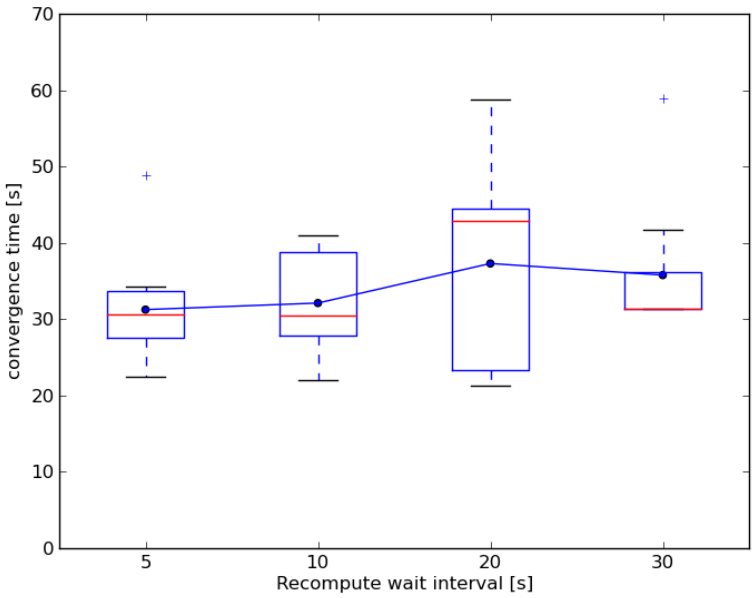


Figure 5.18: Fail-over: 75% SDN, 16 nodes

ployment. However, when the cluster grows the convergence time will increasingly deteriorate.

5.2.3 Announcement

Recompute Wait Interval

Analysing Figure 5.22, Figure 5.23, Figure 5.24, Figure 5.25, Figure 5.26 and Figure 5.27 we can notice that it is hard to see an influence of the RWI value on the routing convergence time.

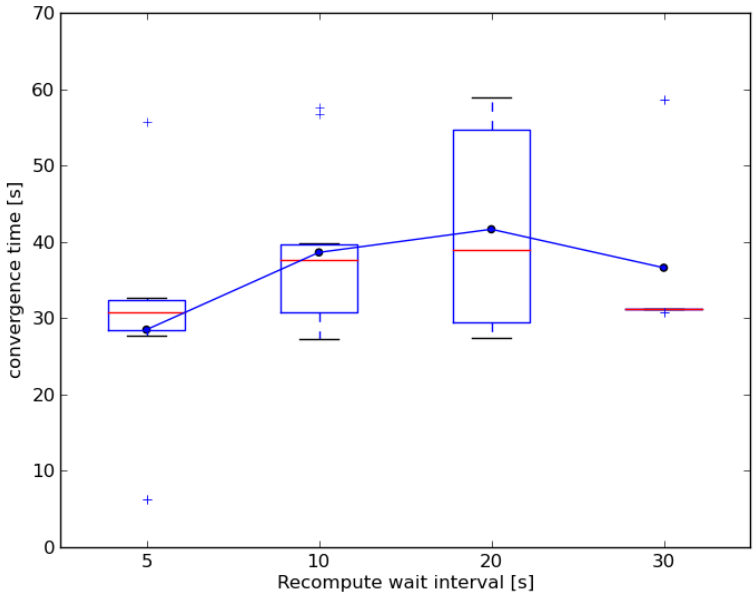


Figure 5.19: Fail-over: 75% SDN, 8 nodes

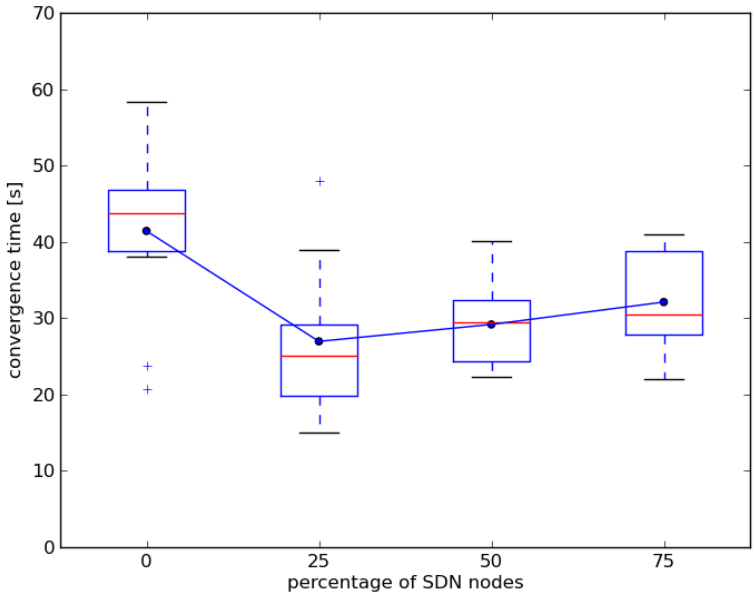


Figure 5.20: Fail-over: 10s RWI, 16 nodes

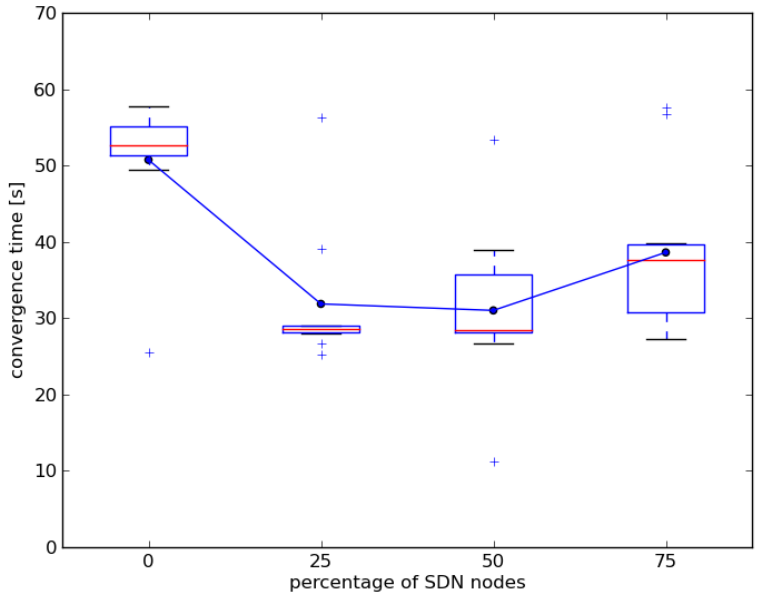


Figure 5.21: Fail-over: 10s RWI, 8 nodes

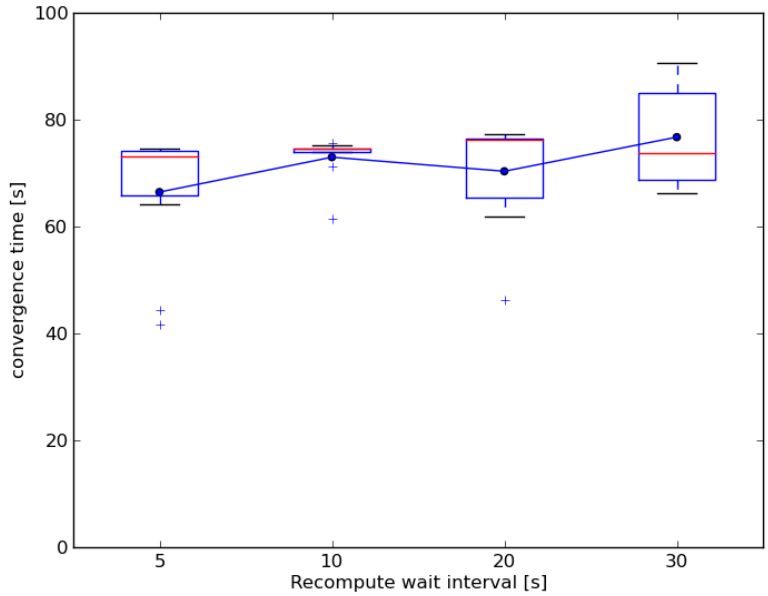


Figure 5.22: Announcement: 25% SDN, 16 nodes

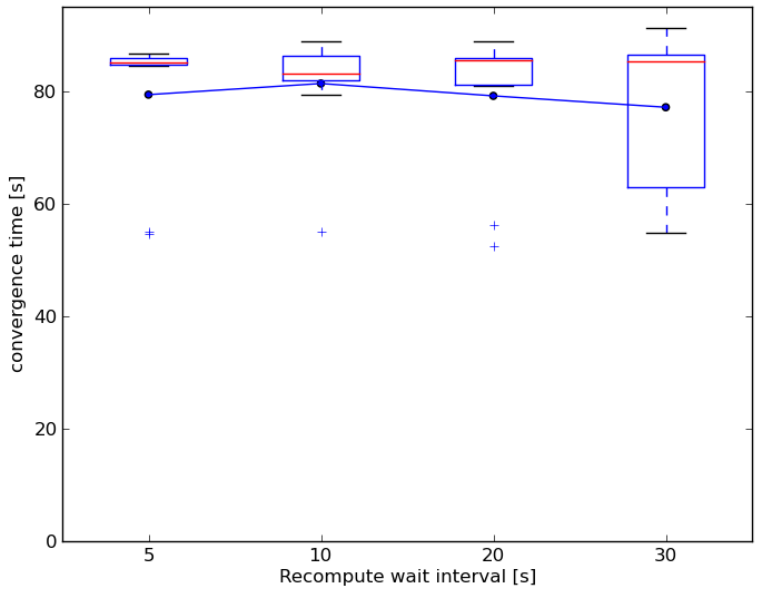


Figure 5.23: Announcement: 25% SDN, 8 nodes

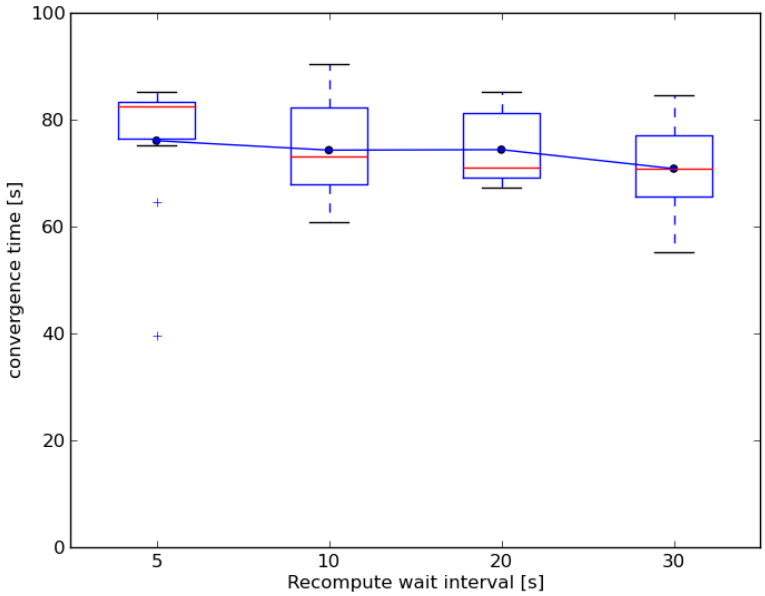


Figure 5.24: Announcement: 50% SDN, 16 nodes

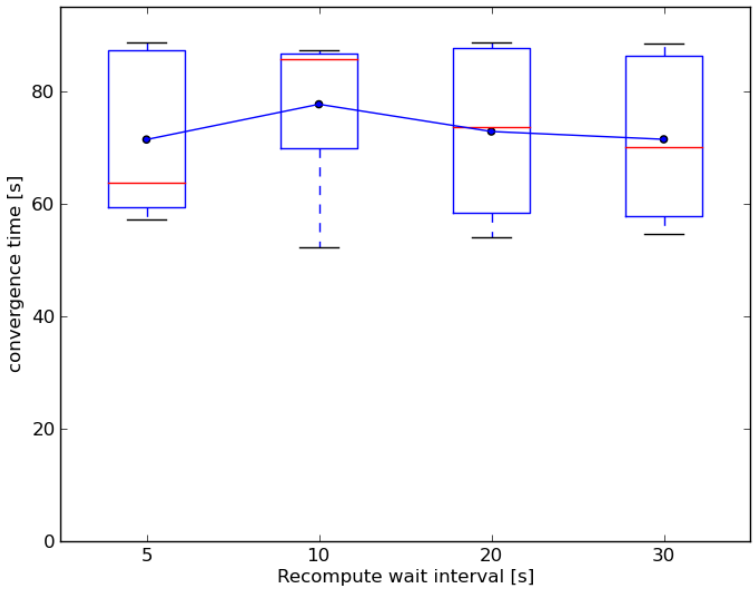


Figure 5.25: Announcement: 50% SDN, 8 nodes

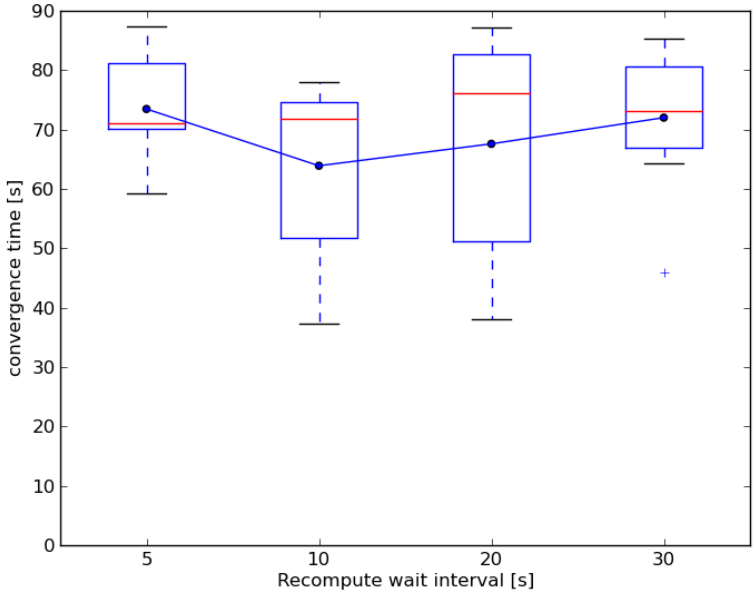


Figure 5.26: Announcement: 75% SDN, 16 nodes

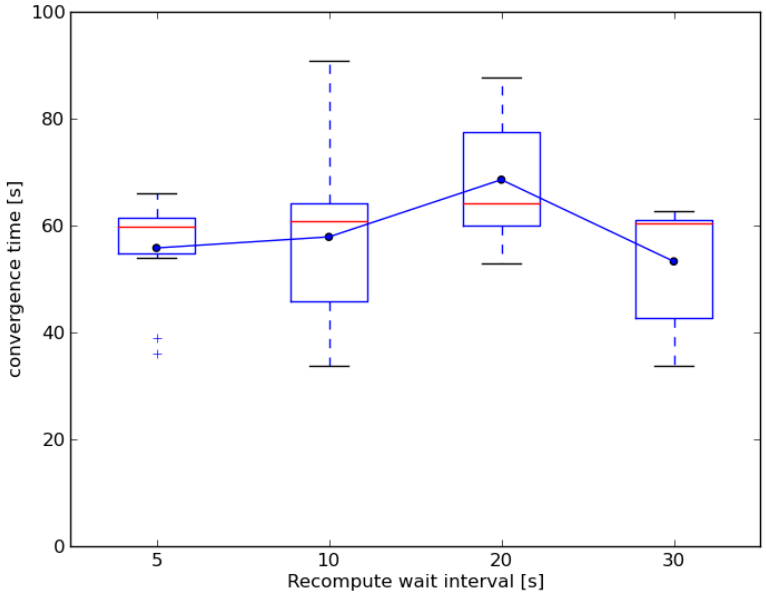


Figure 5.27: Announcement: 75% SDN, 8 nodes

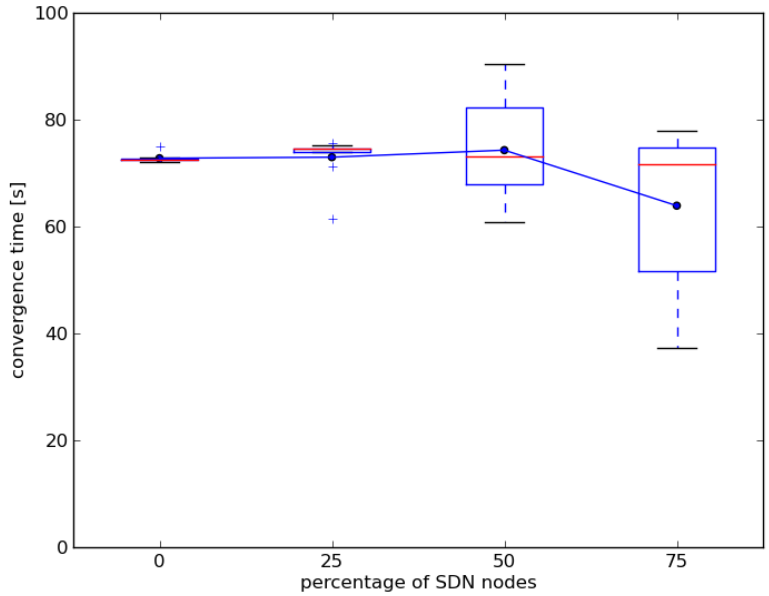


Figure 5.28: Announcement: 10s RWI, 16 nodes

SDN Deployment

When taking a look at the plots for the RWI values 10 (Figure 5.28 and Figure 5.29) and 20 seconds (Figure 5.30 and Figure 5.31) where the percentage of SDN deployment is compared we can see a small decreasing trend the higher the SDN deployment percentage. However there is a difference between 8 and 16 nodes. 16 nodes tend to need a higher percentage of SDN nodes to improve the convergence time. Furthermore we can notice that the range of the convergence time increases with larger SDN deployment. On the other hand Figure 5.32 and Figure 5.33 show that the convergence time can increase slightly with a higher SDN percentage.

Interpretation

We found that with an increasing percentage of SDN switches we can slightly improve the convergence time for certain RWI values. However, to make more detailed statement about the improvements we can achieve we would need to run the experiments more often.
As we think it is a good property that a cluster improves the routing convergence when it grows, we suggest an RWI of 10 or 20 seconds even though the other average convergence times are not much different.

5.2.4 Sources of error

Mininet runs on a single host and every node is a process where they share resources. The effects of running the experiments in such an environment are unknown especially regarding concurrency.
We tried to set up our network in random order. However, we still have some fixed network setup sequences.
The experiments only run 10 times. Therefore the results are not yet very confident.

5.3 Conclusion-Insights

Our measurements suggest that the controller usually improves convergence time but never makes it significantly worse as compared to pure BGP routing. But there are use cases where

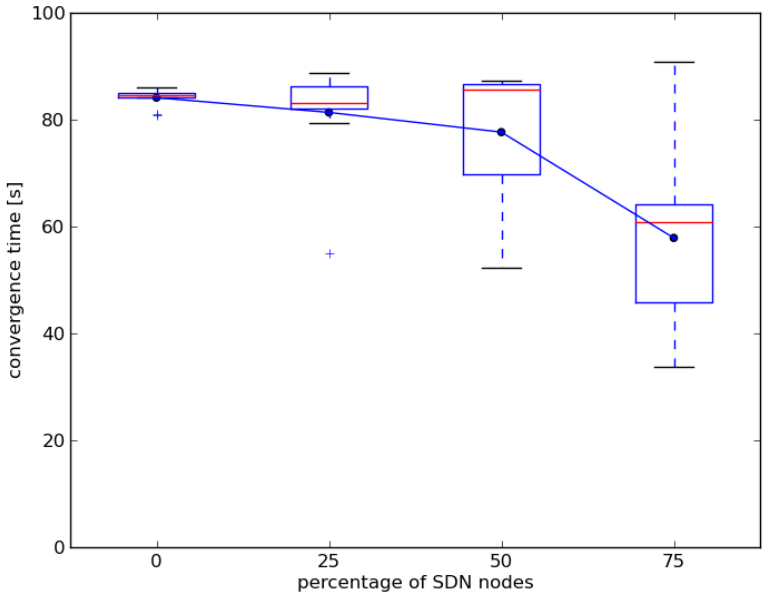


Figure 5.29: Announcement: 10s RWI, 8 nodes

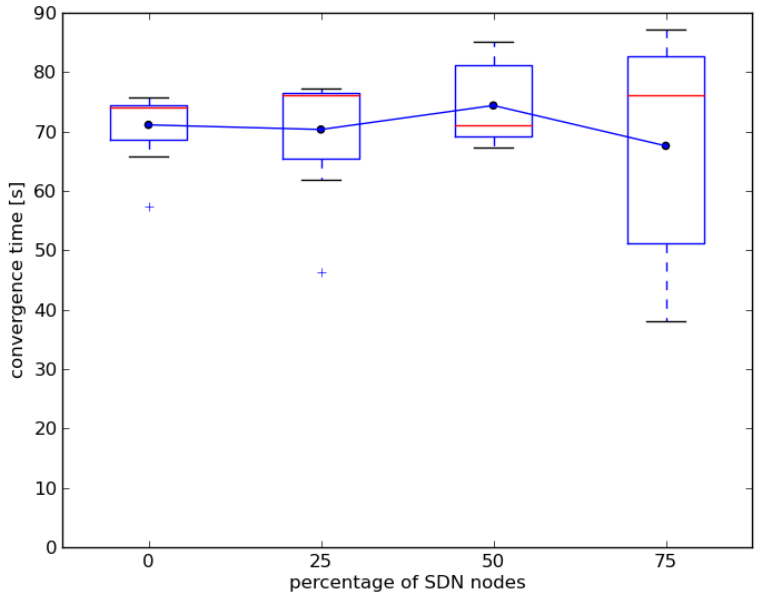


Figure 5.30: Announcement: 20s RWI, 16 nodes

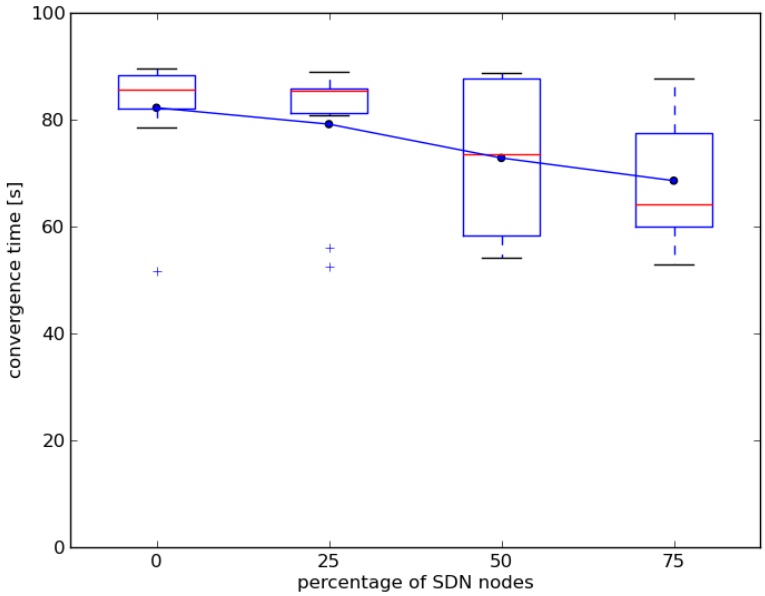


Figure 5.31: Announcement: 20s RWI, 8 nodes

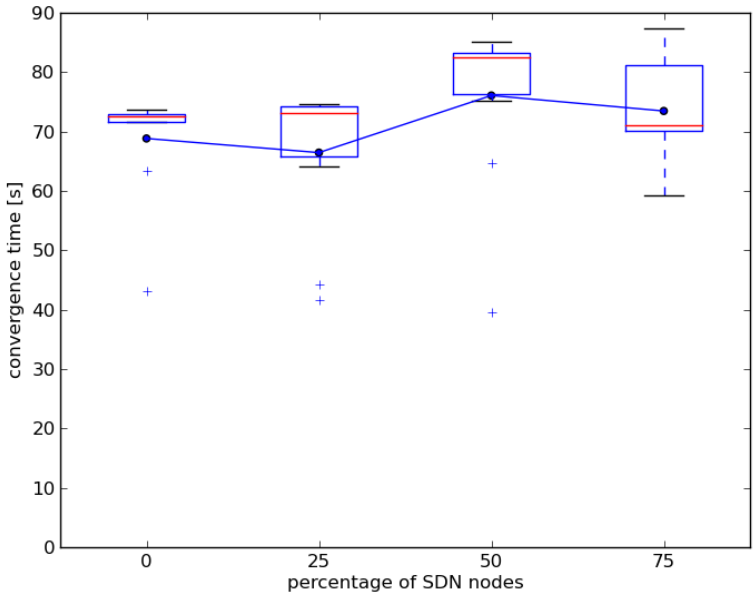


Figure 5.32: Announcement: 5s RWI, 16 nodes

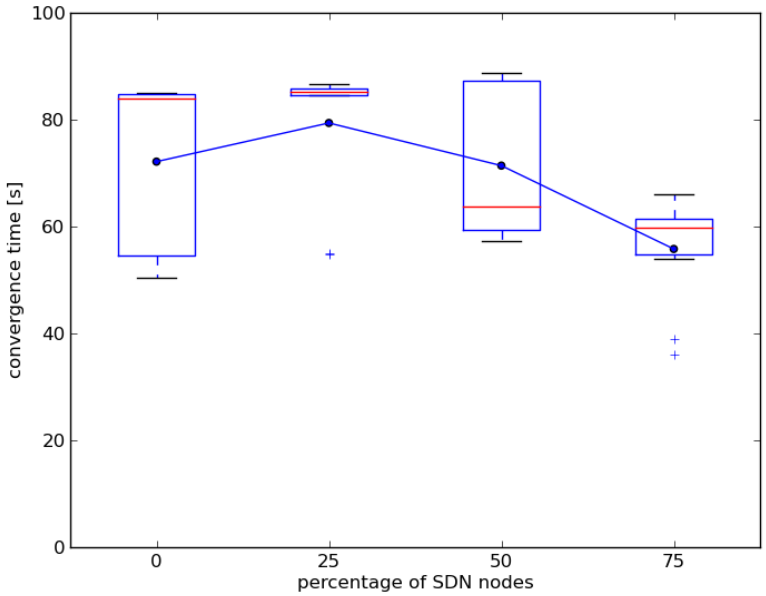


Figure 5.33: Announcement: 5s RWI, 8 nodes

it can reduce the convergence time significantly. Even small SDN deployments can already improve the convergence time.
In the announcement experiment we have seen that the different RWI values do not have a significant impact on the convergence time but we would prefer 10 or 20 seconds because of its decreasing property when the cluster grows. Furthermore we concluded that an RWI of 10 seconds would be best for fail-overs. For withdrawals we have been inconclusive about the best RWI value. We therefore suggest that an 10 seconds RWI is used in general.
As we have seen the convergence time is always lower-bounded by the RWI when at least one SDN switch is involved. This behavior can be easily explained by the design of our controller.
To make more detailed conclusions the experiments have to be run more often. However it is already possible to see trends in our gathered data.
As this analysis is only for clique topologies, they can not be applied to other topologies. Therefore further analysis has to be done on different topologies including more realistic policies. We expect that results on other topologies will verify our observations and amplify the good effect of centralization on IDR routing convergence.

Chapter 6

Future Work

In this section we provide an overview on possible future work.

Policies In the public Internet routing policies are very common and they are also important for convergence, as they define the propagation path of the changes. However the main parts of the topologies are without any policy other than shortest path routing, even though it would be partly supported by our framework. We therefore suggest completing the policy support by implementing policies into the controller, which could be applied to the AS topology graph. We suggest using the patent from Bauer et al [18].

Different topologies Our experiments were limited to the clique topology. We want to extend the experiments to bigger and more realistic topologies with policies, such as topologies build from Internet measurements.

Multiple clusters As IDR clustering would gain adoption clusters would be operated by multiple parties. Therefore there has to be research into the effects of interaction between multiple clusters. Another important point is that SDN clusters operators probably will have different parameters set. For that reason it has to be tested how different parameters interact with each other.
Multiple clusters would also give the opportunity to introduce a new inter-cluster routing protocol, which could be exploited to improve convergence time or to inform neighbor clusters about temporary congestion and negotiate an alternative path.

Stability The current controller path selection does not take into account the already installed path and it is not necessarily deterministic. We therefore suggest performing research into improvements of taking the previous state into account. The idea is that when we have two equal length paths and one path has already been in place, that we do not change to the newer path. We expect to be able to avoid unnecessary route changes and further stabilize the network.

Packet loss Even though our emulation framework has already the ability to measure loss, it is not yet possible to display loss in a appropriate way, e.g. such that it is possible to detect which hosts are losing packets. Furthermore the loss measurements have to be included in the experiments.

Testing Testing our controller application is cumbersome, error-prone and time-consuming. We suggest developing a unit testing framework for inter-domain routing controllers which does not only check for expected flow entry installations but also would interact with BGP. Moreover intuitive installed flow entry analysis tools would be helpful, such as an easy forwarding graph generation for simple flow entries. Furthermore we suggest using an automated controller fuzzing tool.

Effect of intra-domain routing In our experiments intra-domain routing within the Autonomous Systems is neglected. We suggest looking into what effect intra-domain can have on inter-domain routing.

Influence of controller link latency We assumed that the link latency to the controller is zero, but especially in inter-domain routing the routers are distant and latencies are therefore unavoidable. In further work one can investigate the effect of controller link latency on loss.

Different BGP implementations As the BGP routers used in the Internet have different implementations of BGP, we suggest performing research on the effects of these different BGP implementations regarding interaction with our cluster.

Link change Currently it is not possible to take down a link where one of the endpoints is a SDN node. Therefore this can be implemented.

Speed up Starting the network emulation until the network has converged takes some time. We suggest snapshotting the emulator after startup to reuse the initial converged state for multiple different experiments.

Multi-path Multi-path support could be a very interesting extension for the emulator and controller.

Chapter 7

Related work

Prior research on how to improve inter domain routing convergence has been mainly focused on improving BGP or on introducing new distributed protocols.
Kotronis et al [2] then proposed a new solution by exploiting the centralization of SDN beyond AS boundaries and build AS clusters, which was also the motivation for this project.
The Open Network Operating System (ONOS) [19] is a yet to be (publicly) released Operating System for Large Scale Networks. According to the already available public information it is a Distributed Network OS for Large Scale Networks which provides the global network view to its control applications. It is built for scale and fault tolerance. Whereas ONOS is built to be production ready for Large Scale Networks our approach is better suited for prototyping as we for instance use cooperative multitasking. Due to this simplification we can focus more on research questions than on concurrency issues.
RouteFlow [20] is a platform where the controller application mirrors the SDN topology to a virtual network and runs a legacy routing protocol on top of it. Through the OpenFlow controller it installs the forwarding rules from the virtual routers to the SDN switches. Our controller however does not copy routing decisions of legacy protocols but interacts with legacy devices and runs its own algorithm.
There exist several networking emulators. Most of them are focused on either legacy routing protocols or SDN. For example Mininet [11] is a pure SDN emulation framework. On the other hand there exist for example AutoNetkit [21], which provides legacy routing protocols with automatic configuration generation. In contrast to these solutions our emulator framework provides both approaches and can emulate hybrid networks, which allows to run new kinds of experiments. Schlinker et al [22] also developed an emulator based on Mininet which supports SDN and BGP. Unlike our solutions theirs is not focused on building AS clusters.

Chapter 8

Summary

The goal of this master thesis project was to evaluate the effects that a centralized SDN controller can have on inter-domain routing convergence, having the routing decisions of an SDN-capable AS cluster taken in a central component, which also interacts with external legacy equipment over BGP. We designed an SDN IDR controller and developed an emulation framework which supports conducting hybrid BGP and SDN experiments. After having conducted route withdrawal, fail-over and announcement experiments, we concluded that the introduction of AS clusters and logically centralized control can be beneficial to the routing convergence time both within the clusters and without. We suggested using a RWI value of 10 seconds for cluster-controller route recomputation, as an analog to the classic MRAI timer of BGP. Furthermore we found that even small SDN deployments help the network converge faster in certain cases. In route withdrawal experiments the convergence time drops significantly. There is also a small convergence time drop with fail-overs with small SDN deployments. However, route announcement convergence times are only slightly improved in large SDN deployments. Overall, the main results show that BGP convergence time is the upper-bound for the SDN-aided convergence as verified by experiments. Furthermore, some scenarios suggest much better results with SDN achieving reductions up to 85%.

Bibliography

[1] Oliveira, R., Zhang, B., Pei, D., Izhak-Ratzin, R., Zhang, L.: Quantifying path exploration in the internet. In: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement. IMC '06, New York, NY, USA, ACM (2006) 269–282

[2] Kotronis, V., Dimitropoulos, X., Ager, B.: Outsourcing the routing control logic: Better internet routing based on sdn principles. In: Proceedings of the 11th ACM Workshop on Hot Topics in Networks. HotNets-XI, New York, NY, USA, ACM (2012) 55–60

[3] Fu, J., Sjödin, P., Karlsson, G.: Intra-domain routing convergence with centralized control. Computer Networks **53**(18) (2009) 2985 – 2996

[4] Gao, L., Rexford, J.: Stable internet routing without global coordination. IEEE/ACM Trans. Netw. **9**(6) (December 2001) 681–692

[5] Hares, S., Rekhter, Y., Li, T.: A border gateway protocol 4 (bgp-4). (2006)

[6] Cisco: Bgp best path selection algorithm. http://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html Accessed: 19.04.2014.

[7] Pei, D., Zhao, X., Massey, D., Zhang, L.: A study of bgp path vector route looping behavior. In: Distributed Computing Systems, 2004. Proceedings. 24th International Conference on, IEEE (2004) 720–729

[8] : Quagga routing software suite. http://www.nongnu.org/quagga/ Accessed: 18.04.2014.

[9] Rossi, M.: Implementing path-exploration damping in the Quagga Software Routing Suite Version 0.99.13 - patch set version 0.3. Technical Report 090730A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia (30 July 2009)

[10] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review **38**(2) (2008) 69–74

[11] Lantz, B., Heller, B., McKeown, N.: A network in a laptop: Rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. Hotnets-IX, New York, NY, USA, ACM (2010) 19:1–19:6

[12] Wattenhofer, R.: On consistent updates in software defined networks. http://www.csg.ethz.ch/education/lectures/ATCN/hs2013/material/15_SDN_updates_slides (2013) Accessed: 19.04.2014.

[13] : Pox. http://www.noxrepo.org/pox/about-pox/ Accessed: 30.03.2014.

[14] Networks, E.: Exa-networks/exabgp. https://github.com/Exa-Networks/exabgp Accessed: 20.04.2014.

[15] : Open vswitch. http://openvswitch.org/ Accessed: 19.04.2014.

[16] Cooperative Association for Internet Data Analysis: As relationships dataset. http://www.caida.org/data/as-relationships/ Accessed: 20.04.2014.

[17] University of Washington, Computer Science & Engineering: iplane: Datasets. http://iplane.cs.washington.edu/data/data.html Accessed: 20.04.2014.

[18] Bauer, D., Dechouniotis, D., Dimitropoulos, C., Kind, A.: Valley-free shortest path method (March 2011) US Patent 7,907,596.

[19] ON.Lab: What is onos? http://tools.onlab.us/onos.html Accessed: 20.04.2014.

[20] Nascimento, M.R., Rothenberg, C.E., Salvador, M.R., Corrêa, C.N.A., de Lucena, S.C., Magalhães, M.F.: Virtual routers as a service: The routeflow approach leveraging software-defined networks. In: Proceedings of the 6th International Conference on Future Internet Technologies. CFI '11, New York, NY, USA, ACM (2011) 34–37

[21] Knight, S., Jaboldinov, A., Maennel, O., Phillips, I., Roughan, M.: Autonetkit: Simplifying large scale, open-source network experimentation. In: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. SIGCOMM '12, New York, NY, USA, ACM (2012) 97–98

[22] Schlinker, B.C., Zarifis, K., Cunha, I., Feamster, N., Katz-Bassett, E., Yu, M.: Towards impactful routing research: Running your own (emulated) as on the (real) internet. In: Proceedings of the 2013 Workshop on Student Workhop. CoNEXT Student Workhop '13, New York, NY, USA, ACM (2013) 31–34