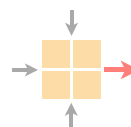




Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



Networked Systems  
ETH Zürich — seit 2015

# An auto-grading framework for the mini-Internet project

Semester Thesis

Author: Martin Vahlensieck

Tutor: Thomas Holterbach

Supervisor: Prof. Dr. Laurent Vanbever

March 2021 to June 2021

## **Abstract**

Since six years the "Communication Networks" course features an hands-on project where students have to configure routers in a larger network and have to establish connectivity. Grading of the resulting work is done manually by only looking at the configuration files and report submitted by the students. Goal of this thesis is to asses the correctness, in an automated fashion, of the routers configuration (done by the students). This can be helpful for grading, maintaining a leaderboard and to give students feedback during the project. In this thesis we show how we achieved this in an automated, transparent, fast and accurate manner. Primarily this is done by testing the live network as configured by the students. At the moment it is checked that each AS makes the correct announcements, correctly implements the BGP business relationships and influences the inbound traffic. One of the created tools already provided feedback to students this year.

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>  | <b>1</b>  |
| 1.1      | Motivation . . . . .   | 1         |
| 1.2      | Task and goals . . . . .                                     | 1         |
| 1.3      | Overview . . . . .   | 2         |
| <b>2</b> | <b>Background and Related Work</b>                           | <b>3</b>  |
| 2.1      | The mini-internet . . . . .                                  | 3         |
| 2.1.1    | Typical topology . . . . .                                   | 3         |
| 2.1.2    | How it works . . . . .                                       | 3         |
| 2.2      | BGP business relationships . . . . .                         | 4         |
| 2.3      | Looking glass . . . . .                                      | 6         |
| <b>3</b> | <b>Design</b>  | <b>7</b>  |
| 3.1      | Looking glass checker . . . . .                              | 7         |
| 3.2      | Isolated AS test . . . . .                                   | 8         |
| 3.2.1    | Testing the BGP configuration . . . . .                      | 8         |
| 3.2.2    | Injecting packets into the network . . . . .                 | 11        |
| <b>4</b> | <b>Implementation</b>  | <b>12</b> |
| 4.1      | Tools . . . . .  | 12        |
| 4.2      | Storing and exchanging data . . . . .                        | 12        |
| 4.3      | Databases used . . . . .                                     | 13        |
| 4.4      | The looking glass checker . . . . .                          | 13        |
| 4.5      | The bgptest container . . . . .                              | 14        |
| 4.6      | files.db . . . . .   | 15        |
| 4.6.1    | Space advantage of storing files inside a database . . . . . | 15        |
| <b>5</b> | <b>Evaluation</b>  | <b>17</b> |
| 5.1      | Grading Model . . . . .                                      | 17        |
| 5.2      | How questions are graded . . . . .                           | 18        |
| 5.2.1    | Question 2.1 . . . . .                                       | 18        |
| 5.2.2    | Question 2.2 . . . . .                                       | 18        |
| 5.2.3    | Question 3.1 . . . . .                                       | 18        |
| 5.2.4    | Question 3.2 . . . . .                                       | 19        |
| 5.2.5    | Question 3.3 . . . . .                                       | 19        |
| 5.2.6    | Question 3.4 . . . . .                                       | 19        |
| 5.3      | Time taken to test an AS . . . . .                           | 19        |
| 5.4      | Comparison with manual grades . . . . .                      | 19        |

|  |           |
|--|-----------|
| <i>CONTENTS</i>                                | iii       |
| 5.4.1 Comparison of points/no points . . . . . | 20        |
| <b>6 Outlook</b>                               | <b>22</b> |
| <b>7 Summary</b>                               | <b>24</b> |

# Chapter 1

## Introduction

For the last six years the "Communication Networks" lecture featured a project, the routing project, to give the students hands-on experience in network configuration. Each group of students would operate their own AS consisting of a few routers and hosts and a few links to other ASes. Together these ASes form the "mini-internet". The students tasks consist of configuring IP addresses and routes, setting up intradomain routing with OSPF and, the main focus of this thesis, interdomain routing with BGP. The task description also tells the students what to put in their report, for example the BGP looking glass of a router or the traceroute between two hosts in the mini-internet. In addition there are also theoretical questions like: Why is it necessary to update the source address of announcements to the loop back address? The duration of the project is four weeks after which, the students submit their report along with their router configuration files. This thesis started out to grade their work automatically, but evolved quickly into a set of tools to verify the correctness of the network, provide feedback to the students and in the future might be used for other tasks like a leaderboard.

### 1.1 Motivation

Before my work, grading looks as follows: After the reports are submitted, the TAs read through it and grade the individual tasks. For tasks that involve configuration, the submitted configuration files are consulted together with information provided by students in their report e.g. a looking glass or a traceroute result. This year there were 36 reports and grading them took a day and involved 4 TAs. The largest change with the auto-grading is that it involves the running and configured mini-internet. Just looking at a configuration file leaves room for error, also some information like the configuration of interfaces with the `ip(8)` command is not included. That makes grading a challenge.

This thesis improves that, by running tests against the live mini-internet while it is running. As an example, the correct filter settings are tested by announcing routes and check how the propagate through the AS, or to see if students really manage to influence inbound traffic the grader uses a simplified implementation of the BGP routing decision engine.

### 1.2 Task and goals

Goal was to automate in part the grading process of the project. This is achieved by disconnecting ASes from the mini-internet and connect them to a tester which then tests BGP route propagation and connectivity.

As a side effect another tool was created, the Looking Glass Checker, which in contrast to the other test does not actively manipulate the network. The looking glass checker gets its information from the looking glasses of the routers. Its output was made available to students already this year, and they could use the feedback to fix the errors in their BGP configuration.

The created grader also makes it possible to do grading during the project itself and can be another way to give feedback to students.

## 1.3 Overview

Section 2 gives a bit of background on the thesis, section 3 talks about the high level design of the tools. In section 4 the implementation is presented and in section 5 how the configuration of the students is evaluated. Finally there is an outlook in section 6 and a short summary in section 7

## Chapter 2

# Background and Related Work

This chapter provides a brief overview of what is a mini-internet, how a typical mini-internet looks like and how it is implemented. Also there is a quick overview of BGP business relationships as used in the mini-internet and short description what a looking glass is.

### 2.1 The mini-internet

A mini-internet is a virtual network to give students hands-on experience by having them establish network wide connectivity using OSPF and BGP.

#### 2.1.1 Typical topology

Our mini-internet consists of ASes and IXPs. Each AS in turn consists of routers and hosts. Some hosts form one or more layer 2 network where students have to set up correct VLAN configuration to establish connectivity, the other are connected to routers. Figure 2.1 shows how the network looked this year.

In that figure you can see the connections of the AS to other ASes and the IXP (in light blue). There are two relationships between ASes: Peer-Peer or Provider-Customer. These relationships define which routes are announced (see below).

An AS that is not a customer to any other AS is a tier 1 AS, one that is not a provider to any other AS is a tier 3 AS, the rest is tier 2. Students are in charge of tier 2 networks. The mini-internet can be divided into logical regions, where a region consists of tier 1, tier 2 and tier 3 ASes. Figure 2.2 you can see how the individual ASes were connected this year. The beige rectangles numbered 0-5 are so-called regions. Notice that in this topology a region is the set of ASes that can be reached by only crossing Provider-Customer links.

In total there are 78 ASes and 7 IXPs.

#### 2.1.2 How it works

Each router, host and switch in the mini-internet is a docker container with the appropriate network interfaces connecting them to other containers. While it seems that these connections are direct, in reality it looks like in figure 2.3. In this example the PARI routers (assumed to represent a router in Paris) of AS 3 and AS 4 seem to be connected via the dotted line, in reality all their traffic passes through the host. This design allows us easily to temporarily reconnect e.g. PARI of AS 3 to a special container of our own and back afterwards.

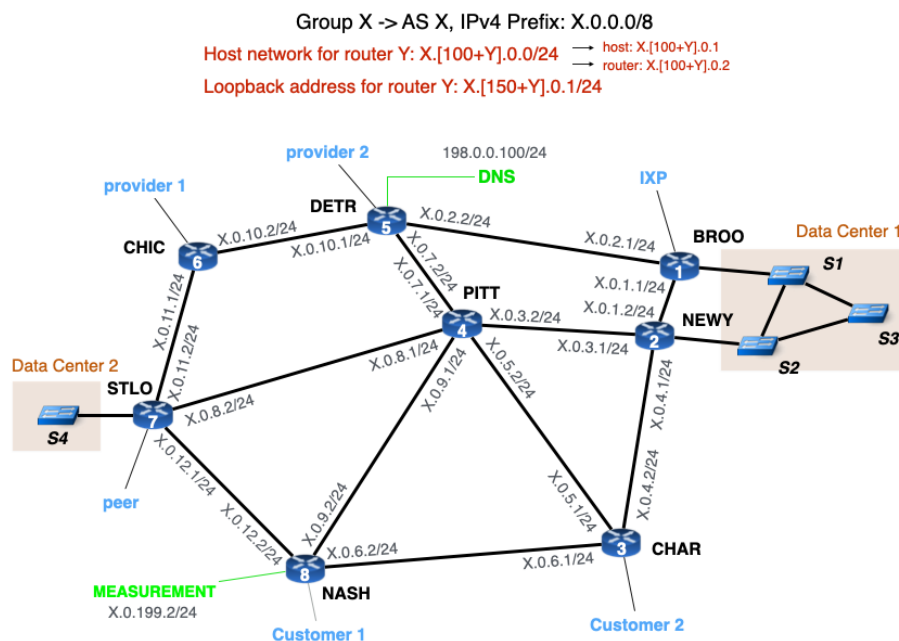


Figure 2.1: Inside an AS, this is taken from the 2021 task assignment

are exported to:

| routes from ... | Customers | Peers | Providers |
|-----------------|-----------|-------|-----------|
| Customers       | Yes       | Yes   | Yes       |
| Peers           | Yes       | No    | No        |
| Providers       | Yes       | No    | No        |

Table 2.1: BGP business relationships

The students have SSH access to all the containers of their AS and the relevant software is already installed.

## 2.2 BGP business relationships

Each connected pair of ASes follows a business relationship. Either they are peering, in that case this is a Peer-Peer relationship. Otherwise one of them acts as a provider, providing services to the other AS, the customer. Customers pay providers money for their traffic they send and receive from them. Peers do not pay peers. For these reasons ASes must choose which prefixes to advertise to their neighbors. Getting paid for all the traffic, provider naturally announce all prefixes they know to their customers. To their providers they only announce their own customers (as they have to pay the provider but eventually get paid by the customer). To peers ASes only announce their own customers, everything else would only cost money. These relationships can be seen in table 2.1. Each row names the business relationship with the AS the routes are received from, each column is the business relationship with an AS routes are potentially send to. Yes means the route is propagated, no means to route is not announced. For more information about this model see this<sup>1</sup> paper.

<sup>1</sup><https://people.eecs.berkeley.edu/~sylvia/cs268-2019/papers/gao-rexford.pdf> (7.6.2021)



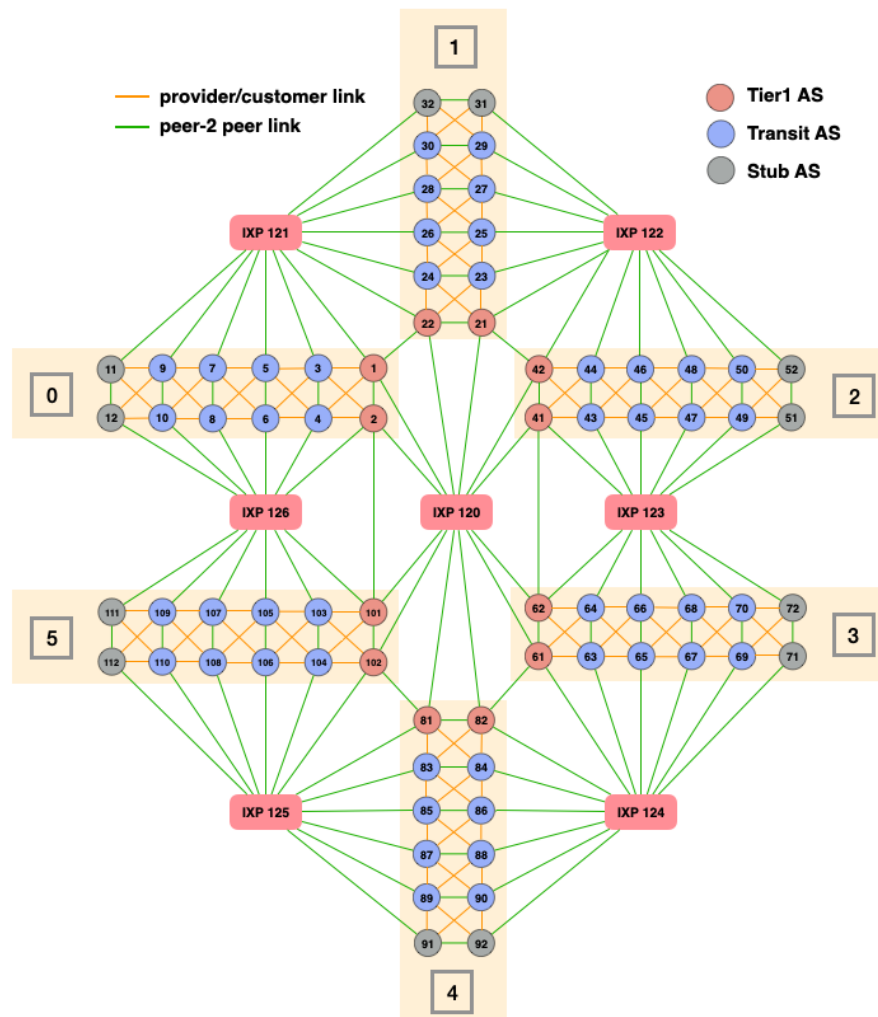


Figure 2.2: The mini-internet, how it looked this year, taken from this years assignment

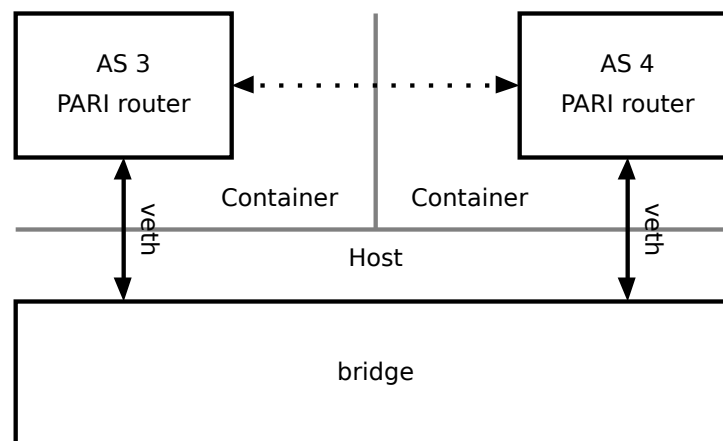


Figure 2.3: The two containers of AS 3 and 4 think they are connected via the dotted line but in reality they are bridged via the host

A BGP leak occurs, if IP addresses are not announced according to above rules. There are also other types of BGP leaks, like for example announcing IP addresses used internally between routers.

## 2.3 Looking glass

A looking glass gives information which prefixes can be reached. When there are multiple routes to the same prefix it is indicated which route is preferred. Another property is the AS path, which is a list of ASes through which this announcement was passed and which send packets will follow. In our mini-internet they are generated and saved automatically for each router.

# Chapter 3

## Design

The created framework consists of a looking glass checker (verifying that the BGP business relationships are honored, already deployed this year), the isolation tests (putting an AS under the microscope). In addition there exist various little helper tools to support and supplement these three tools.

### 3.1 Looking glass checker

The task of the looking glass checker is to find policy misconfigurations. It does so by solely looking only at the looking glasses and the network topology. This "non-invasive" property, i.e. only exported data is used and there are no changes to the topology of the mini-internet, makes this tool quite easy and safe to use. To demonstrate that: It was possible for me to analyse the running mini-internet from a different computer by simply downloading and arranging the published looking glasses correctly.

The looking glass checker tries to detect the following misconfigurations:

- Announcements which do not follow the business relationships e.g. exporting routes from a peer to a provider.
- Missing announcements, found by analysing the topology.
- Checking that in theory there is no better route to a prefix (better here means using Customers over Peers over Providers).
- Route leaks via IXPs to ASes in the same region.

The problem with this approach is, that it only considers what actually makes it into the looking glass. This can lead to the wrong impression of correctness. Here are two examples:

- If incorrect routes are blocked on the receiving side rather than on the sending site (where they should be blocked), it still gives the impression that the business relationships are correctly implemented. See figure 3.1 for an example. In both cases the looking glass checker won't report an error.
- In the same setting as above, assume that AS 4 is not exporting anything (e.g. due to inactivity). As there are no routes received by AS 3, it is impossible to tell if it has configured its policies correctly for the peer (as there are no routes to wrongly distribute)

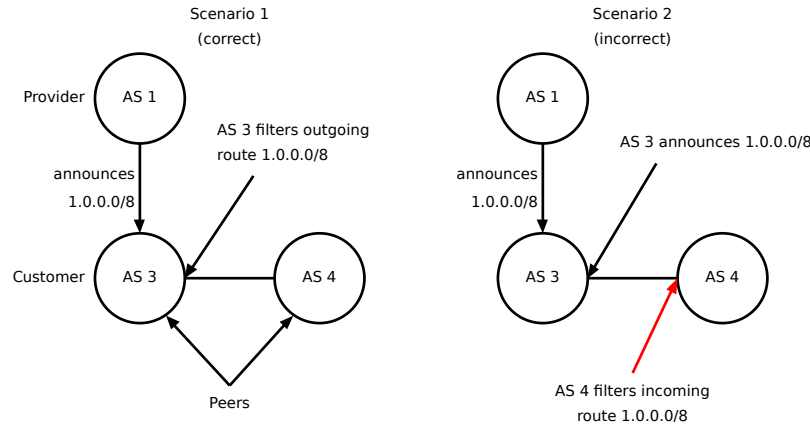


Figure 3.1: On the left AS 3 correctly blocks the outgoing announcement to AS 4. On the right AS 3 wrongly announces a provider route to a peer but AS 4 blocks the announcement.

Because this tool was finished early in the thesis, and due to its non-invasive nature, it was already used this year to provide feedback to students. A selected subset of errors was uploaded to a website for the students to inspect reports related to their AS. After the project was over, only two ASes were still listed there.

The first errors the looking glass checker found in the real mini-internet were the following:

- Tier 3 AS exported their prefixes to tier 1 ASes via an IXP
- A misconfiguration in the TA controlled AS 109

## 3.2 Isolated AS test

As it turns out, the shortcomings of the looking glass checker can be easily addressed by isolating the AS. Isolating an AS means disconnecting the running AS from the running mini-internet and connecting it to a special test container (the bgptest container). To illustrate this process, see figure 3.2, which shows the external connections of AS 5 before and after isolation. It is important to mention: The test container imitates IP addresses and AS numbers of the would-be peers, no configuration changes are made inside the AS that is being isolated. After the old eBGP sessions expire (with the peers in the mini-internet) the routers will simply establish new ones with the test container. To speed up the process we manually issue a command to renew the sessions. With this setup we can do two things: Make crafted announcements to expose policy errors and send traffic into the network.

### 3.2.1 Testing the BGP configuration

In this part we make use of the ability to create arbitrary announcement to test the BGP configuration of the isolated AS. This includes:

- Find announcements to verify that the filtering is done in a robust way
- Check that an announcement is blocked (this is a task of the students)
- Check that some announcements have the correct community values set. This is done with the isolation tests because it is very easy to do.

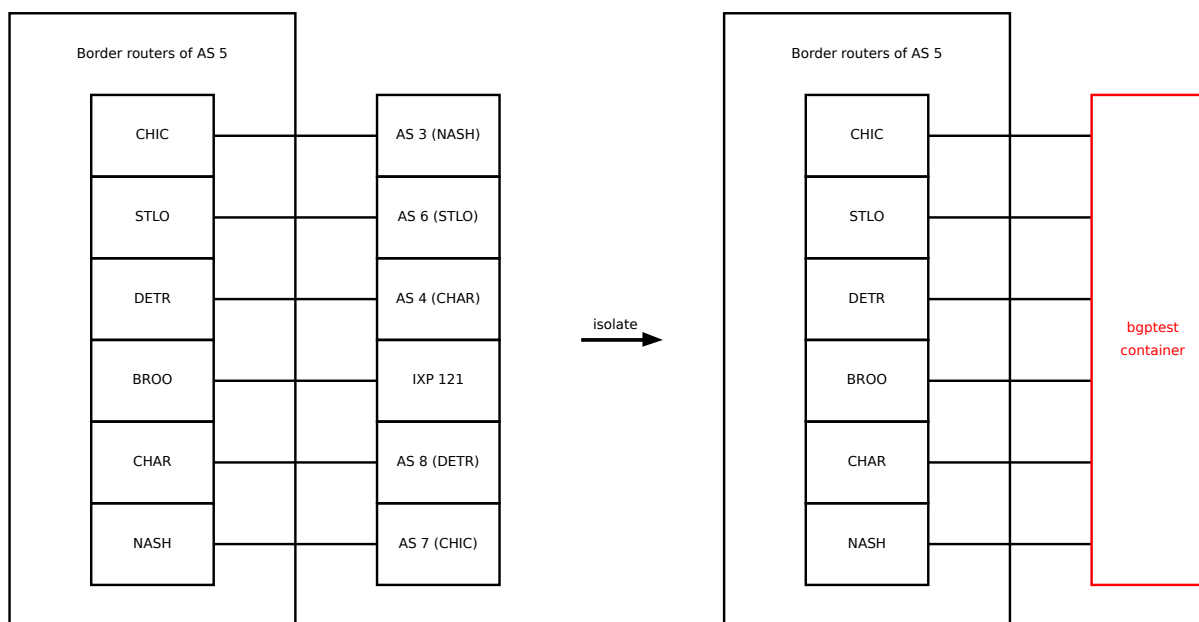


Figure 3.2: Left: External connections of AS 5 in the mini-internet. Right: External connections of AS 5 while being tests

A huge advantage over the looking glass checker is that BGP configuration errors cannot be hidden by the configuration of other ASes.

The easiest thing to check is the third point. Here students need to set community values depending on their AS number and the AS numbers of the ASes on the other side of the IXP (the ones in the other region). Community values are attributes of announcements that can be chosen arbitrarily. Depending on the community value, the IXPs propagate (or not propagate) the announcements to the other ASes they are connected to. This mechanism is used so that ASes in different regions can peer over an IXP. Checking that an AS is setting the correct values is trivial with this setup, as they can be directly read from the looking glasses in the bgptest container. In contrast the looking glass checker only checks that no announcements are made to ASes in the same region and even for that depends on missing input filtering at another AS. This way we check that exactly the correct community values are set, no more, no less (depending on how students use community values inside their AS without clearing them at the border, the check might be relaxed to: all expected values are present and all forbidden values are not present).

### How filtering should (not) be done

The intended idea is that students mark every incoming routes in a way to signify where it was originally received from (e.g. if AS 3 learns a route from its provider it marks it as "learned from a provider", or "learned from AS 1"). This knowledge is then used in output filters, where one can say "only export routes which were learned from a customer"). But filters are quite flexible and can also match on other properties like the community values of the route, the prefix or the AS path.

We try to find configurations hard-coding prefixes and relying on AS paths in unsafe ways. If students allow or block specific prefixes, their configuration will break when an announcement with a new prefix arrives. Relying on the AS path attribute without validation is dangerous, as neighboring ASes can modify it quite easily.

```

NASH: prefix 201.15.0.0/16, path:
NASH: prefix 201.25.0.0/16, path: 5
NASH: prefix 201.35.0.0/16, path: 500
STLO: prefix 201.16.0.0/16, path:
STLO: prefix 201.26.0.0/16, path: 4
STLO: prefix 201.36.0.0/16, path: 500

```

Figure 3.3: An excerpt of fake announcements made to AS 3

## Announcements

In this section I present the announcements that are made to the target AS.

First every session pretending to be a neighbouring AS announces its AS specific prefix (in the scenario of figure 3.2 the bgptest container would announce 3.0.0.0/8 to the router CHIC).

With the following announcements we try to find the configuration errors mentioned in the last section. We do that by announcing three additional routes to each border router. Each of these announcements has a different prefix which is unique to the router to which it is announced. Further it is a prefix which is not used in the real mini-internet. Each of the three announcements has a different path:

- The empty path
- The path a real announcement would have (except in case of the IXP)
- The path 500 (an AS number not used in the mini-internet)

To the router connected to the IXP receives one additional announcement to test another task where students need to block that announcement using filters. An excerpt of announcements made to the Nashville and St. Louis routers is shown in figure 3.3.

Here is a list of errors we try to detect and how they are detected:

- The students block outgoing announcements based on the AS path → the announcements with the empty and 500 AS path will still make it through and trigger an error.
- The students allow announcements based on the AS path → it will trigger an error if the routes with the fake AS path are not propagated correctly.
- The students rely on the prefix → will trigger an error because either they are not correctly announced (if they have hard-coded a list of "good" prefixes) or not correctly blocked (because they have hard-coded a list of "bad" prefixes)
- The special announcement made to the IXP router is blocked → this is tested by checking that this announcement is not exported anywhere. But this is not conclusive: The AS might be accepting the announcement and is simply not exporting it to its neighbors. To find that case additional checks must be performed like checking the internal looking glass of the router connected to the IXP.

### 3.2.2 Injecting packets into the network

By injecting packets into the network we can verify connectivity inside the network (e.g. the hosts connected to the routers are accessible) and between the border routers. As we only can inject/receive packets at the border there are roughly two types of tests we can do:

- We can send packets through the network from one edge to another.
- We can send packets to routers/hosts inside the network and rely on ICMP messages to inform us of success/failure.

Note that these tests require the BGP announcements we make and BGP to work to a certain degree in the tested AS.

To distinguish packets and responses, we add a unique payload to each of them.

#### Edge to edge

For the first type we use UDP packets. As source/destination IP we select an IP inside the prefixes which are announced at the source/destination router.

The following connections are tested: We test that Customers can reach Providers and the other way around. Next we test the connection between Peers and Customers and between Peers and Providers. You will notice that in theory the latest case should not happen. No traffic should ever flow from a peer to a provider or the other way around. This is a violation of the business relationships (the relaying AS will accept/send the traffic for free from the peer and pay the provider for sending/receiving traffic from it). Presence of this test is simply due to a coding error and nicely demonstrates one of the problems in the internet.

#### Testing hosts inside the network

Some hosts inside the network do not have a direct connection to the bgptest container, so the previous method does not work. Therefore the second type uses two (actually three) types of ICMP messages.

- We ping the hosts using ICMP echo requests and looking for the corresponding echo reply, and
- we perform a traceroute looking for time exceeded messages triggered by packets with too low TTL values.

The results of these tests are not used yet. In the future they could be useful to provide feedback to students during the first weeks where they have to configure the internals of the network, but that would require some changes, because at the moment we rely on the fact that BGP is configured in the target network (which is done after the internal routing works in the routing project). The traceroute and the ping might seem redundant but the path information of the traceroute might be used to verify aspects like the correct setting of OSPF weights.

Another improvement would be the ability to inject packets at the hosts and by testing with connection based protocols like TCP.

## Chapter 4

# Implementation

This chapter discusses various topics related to the implementation. First there is an overview of the tools/software used in the project. Next there is a small section about how databases and SQL are used for various tasks and a list of databases which are used. Following that is an introduction of the bgptest container which interacts with the AS during the isolation tests. Finally there is an overview of the concept behind the current grader and a short tour how the individual questions are graded.

### 4.1 Tools

This is a list of tools/software used throughout the project.

- Bash: Used for most tasks, especially ones which need to call a lot of commands.
- Python 3: Used for tasks which are too complicated in Bash.
- Docker: Used to run the mini-internet and test containers.
- OpenVSwitch: Used to connect container via host bridges.
- Go: Only used once, mainly for the easy concurrency it offers.
- SQLite 3: Databases. Used in shell scripts with `sqlite3(1)`. Python 3 can access SQLite 3 databases with the standard library.
- Git: Version Control, used for mini-internet and this project.

I also had access to two virtual machines to run smaller versions of the mini-internet. The largest mini-internets I used were 20 AS with 3 IXPs.

### 4.2 Storing and exchanging data

The created software makes heavy use of databases to store and exchange data. This was not the plan at first, but rather happened "by accident" while I was writing the looking glass checker, and was in the need to structure data into an easily queryable way. It turned out to be a very convenient way of doing things and was in part inspired by the texts on the SQLite web page <sup>1</sup> and how fossil<sup>2</sup>

---

<sup>1</sup><https://sqlite.org> (7.6.2021)

<sup>2</sup><https://fossil-scm.org> (7.6.2021)



works. At one point it might have even advanced a bit too much just as an experiment to test out the limitations of this approach. While it adds an dependency (SQLite3) to the project, I think this is a warranted one and doesn't unnecessarily increase setup complexity being widely available. SQLite3 is widely available (it is in the public domain), and bindings exist to many programming languages, and are in Python 3 standard library.

SQL, while also used with the classical selects, also features slightly more complicated queries, for example generating a recursive list of providers, finding the highest numbered AS in the region connected to the same IXP or splitting a list of integers at gaps in the sequence.

The biggest drawback are, that at least a basic understanding of SQL is required, and that it would be quite an undertaking to get rid of it again.

### 4.3 Databases used

- `as.db`: Contains information about the topology and parsed routing tables and is used to implement the looking glass checker.
- `bgp.db`: Contains aggregated information from the ovs and links database.
- `config.db`: Contains various information about the mini-internet topology. It creates this information by parsing the configuration files of the mini-internet.
- `files.db`: Contains the output of various commands executed inside the mini-internet containers.
- `links.db`: Contains information about selected network interfaces. Its input is parsed from the `ip(8)` command.
- `ovs.db`: Contains information about the ovs bridges and ports. Generated by parsing various `ovs-vsctl(8)` output.
- `results*.db`: Contains various information for the scripts running inside the BGP test container and the results of these tests. There exists one for every tested AS.

In the future some of them might be consolidated.

### 4.4 The looking glass checker

The looking glass checker consists of three files: `cfparse.py`, which parses the relevant part of the mini-internet configuration and stores it in `as.db`, `lgparse.py` which reads the looking glasses into the database and `lganalyze.py` which checks the looking glasses for errors using the topology information.

All of the information needed is also present in `config.db`. The decision was made to keep using `as.db` in order that the looking glass checker is a standalone tool.

`lganalyze.py` includes several checks. Some of these were made available to students already this year. These can be found by looking for `ERROR-SIMPLE` in the source code. More error codes exist, these can be found in the source code. Here a quick overview of the checks and how they are done:

**Business relationships** This is done by following the AS path of each route and check the relationship between the ASes of the links crossed. Implemented as a state machine. As a single invalid export might cause another error every time the route is propagated, all errors after the second link is crossed are treated as NON-LOCAL errors, which should have been already detected.

The local errors are visible to students.

**eBGP IP leaks** eBGP sessions use IPs in the range 179/8 (between ASes) and 180/8 (between IXPs and ASes). This is done by simply checking for these prefixes (actually all prefixes which look unexpected)

**All announcements** Checks that every AS announces all the prefixes it should. What should be announced is decided by looking at the topology.

**Best paths** Verifies that indeed the best path is used. The best path is marked in the looking glass. It is checked that in theory there is no better path, better here means prefer paths via customers over paths via peers over path via providers. In the future this might be changed to consider that the best path given the announcements is used.

**IXP handling** This code is responsible to check that no prefixes are announced via the IXP to ASes in the same region.

The errors are visible to the students.

The errors are all collected into a database to prevent duplicates. In the default mode these are printed to stderr. For the routing project an additional mode was added, where errors are limited to ERROR-SIMPLE errors and to tier 2 ASes and the output is directly formatted as HTML, so that it can be directly uploaded to a web page. Sometimes the same errors are saved under different categories e.g. to have a more verbose or detailed statement for TAs or analysis.

## 4.5 The bgptest container

The bgptest container is one of the most important tools of the grader. By using `ip(8)` and `ovs-vsctl(8)` the edge ports of an running AS are disconnected from the mini-internet and connected to a dedicated docker container (called the bgptest container). The Python test scripts and results database are then copied into the container and executed. After that the results are copied back out and the AS is disconnected from the test container and connected back into the mini-internet.

To be able to interact properly with the network all the links inside the bgptest container are configured to simulate the respective neighbor of that AS (in terms of IP address). Inside the bgptest container `exabgp` is used to maintain a eBGP session with all border routers of the AS. `Exabgp` is used because it is possible to maintain multiple sessions imitating a different AS in each and because it is very easy to announce routes. By having the ability to do arbitrary announcements it is very easy to test the bgp policies because the inputs are well defined and can try to test potential corner cases.

In the test script inside the container, first it makes various bgp announcements. After that it waits a moment for BGP to converge and captures the resulting `exabgp` looking glass. Scapy is then used to send different packets through the network and to different network locations. These packets are then captured again when leaving the network and saved in the results database along with the packets which were send.

To identify received packets they are given a unique<sup>3</sup> payload. With this payload it is possible to identify the origin and intent of a packet. It is a comma separated list of the following values: It is a comma separated list of the following values:

- Identifier same for the entire test. Consists of a 4 byte test number, 5 byte timestamp and 7 random bytes hex encoded.
- Type of the packet, see below
- Source location
- Source IP address
- Destination location
- Destination IP address
- The number of the packet. Each packet is sent three times.
- (Optional) TTL value in the IP header (used for traceroute packets)

## 4.6 files.db

Inside this database information is extracted by executing commands in the docker containers. Collected information includes:

- IP address and routing information extracted with ip(8) from the host and layer2 containers.
- Looking glass of IXPs (text form)
- Looking glass in text and json, and FRR configuration from routers.

In the future it might even be possible to restore the mini-internet configuration by using these files. They are saved inside a database for ease of use (access and copying).

### 4.6.1 Space advantage of storing files inside a database

Recalling a post about saving disk space by storing files in a database rather than the file system<sup>4</sup>, I moved for a little experiment.

The following test was conducted using the files.db database from this years routing project. The individual files were unpacked from the database into a directory called output.

```
$ du -hd1 output
82M      output
$ du -hd1 files.db
76M      files.db
```

This corresponds to a saving of ca. 7%. The files are not explicitly compressed.

Now try with compression. Every file inside output is compressed with gzip. These files are also inserted into a database files2.db.

<sup>3</sup>At the moment it is still very easy to accidentally get the same payload twice. This should definitely be changed in the next version

<sup>4</sup><https://sqlite.org/fasterthanfs.html#summary> (25.5.2021)

```
$ du -hd1 files2.db
4.2M    files2.db
$ du -hd1 output
14M     output
```

Database is 70% smaller.

Now lets compress the directory of uncompressed files with

```
$ tar czf output.tar.gz output
$ du -hd1 output.tar.gz
3.4M    output.tar.gz
```

Despite this I would keep the files uncompressed for ease of access of other tools and with sqlite3(1).

# Chapter 5

## Evaluation

### 5.1 Grading Model

The model is basically pile of files. Most of the scripts follow the UNIX approach and only do one thing. As an example the only tool doing any grading at the moment is the grader itself, using data collected by other scripts. This has several benefits:

- It keeps the data in its raw form as long as possible, making it easier to adjust the grading afterwards. As an example: At first the bgptest container ran tests and assigned points given for each. Should now afterwards a bug in the grading code be discovered, it would be hard to change that. In the current scenario the grader could just be modified and executed again.
- Adding new source material is easy: Just put it in an database and modify the grader to use it. New code would not need to know anything about grading (e.g. just execute traceroute in docker and save the results and the grader will parse them)
- It is easy to use the gathered material in other ways, e.g. the send and received packet counts could help to detect malfunctions inside an AS.

At the moment the grading code has tables to keep track of manual points as well. It was thought as an experiment to see how it would work to keep the entire grading in one database. As a proof of concept I also wrote a little shell to interact with the database in simple ways. Looking back at it, it seems to be a bit of an overkill, and the grader should just do one thing: spit the points out (in a database or otherwise) with the notes explaining its decision/giving hints at strange configurations etc.

#### Note

While writing the report I found two issues with the grader:

- The grader expects the rogue announcement from the stub AS to show in question 3.1. This assumption is wrong, students might have already blocked the announcement, this led to them getting no points.
- There is no code in the grader for task 3.2 to check that the rogue announcement is correctly blocked.

For the comparison with the grades awarded by the TAs I have fixed the first issue by simply giving points with and without the rogue announcement. Note that this was very easy due to the nature of the grader (being run offline and not requiring the mini-internet to be running). The second issue would be a bit trickier to fix, I haven't done so because of the remaining time. While it would work for most ASes to simply see if the rogue announcement makes it to the customers this does not work for the providers of the rogue AS, they will announce the prefix to the customers due to the customer to customer business policy. Anyways this would not be correct, as an AS might be blocking the rogue announcement to leave the AS, but uses it in its internal routing tables. The correct way to fix this will probably require using the internal looking glasses of the AS. Again, changing this will not be impossible, we have saved all the looking glasses in files.db. In this case, the looking glass checker would catch the misconfiguration quite easily (but report it as an error of the stub AS and the students wouldn't see that).

## 5.2 How questions are graded

The grader cannot grade all questions, at least yet, and even the ones it can grade it is mostly only possible in part. This is due that most question also include some tasks like to describe something or insert a snippet of a looking glass. What the grader does grade are the implementation parts of the tasks like announce your network or implement the business relationships. Each question is graded in its own function and receives three arguments: A dictionary of the available databases, the number of the AS that should be graded and the maximum number of points achievable.

Below is a quick summary of interesting/important points in the implementation of each question that is graded.

### 5.2.1 Question 2.1

For this question, only the collected configuration files of the routers is used. The students task was to implement a full-mesh configuration with update-source. Largest part of this function is a rudimentary parser of FRR configuration files, Which checks that the expected directives are present. While writing this I see it does only checks for the presence of ONE iBGP connection, not all of them.

### 5.2.2 Question 2.2

Task was to announce the own network. At each neighbor all prefixes belonging to the tested AS are combined and it is checked that these prefixes sum up to the assigned /8 network. This combination of prefixes is done to allow groups using more specific prefixes in locations to change the inbound traffic or preventing the hijack.

### 5.2.3 Question 3.1

In this task, the students have to configure their filters such that routes are distributed according to business relationships. In contrast to the looking glass checker, this simply computes the set of expected routes (this is easy because we control all announcements) and then compares them to the actual routes received.

#### 5.2.4 Question 3.2

Here students were asked to add community values to the routes announced to the IXP in order to have their routes distributed to the ASes in the adjacent region. The implementation features a recursive SQL statement to compute the list of AS in the same region. At the moment it is expected that all routes have the same community values.

#### 5.2.5 Question 3.3

This task required the students to influence inbound traffic to prefer the Chicago router over the Detroit router. To verify this the grader feeds the received routes from Chicago and Detroit to a `simple_best_route` function. It is called simple, because it only implements the sensible subset of steps in the routing decision process. Sensible means it only considers points students can configure and rely on not things like e.g. what is the IP of the eBGP session.

#### 5.2.6 Question 3.4

Goal of this question was to prevent part of the prefix being hijacked by another AS. The grader checks that there are more specific announcements and that they cover the hijacked IP space.

### 5.3 Time taken to test an AS

When run locally the time to test an single AS with the isolation tests takes 2 minutes. After the routing project ended we had the opportunity to test the grader directly on this years mini-internet. This took much longer than on the test systems, around 20 minutes. Still it is acceptable to run such tests overnight in the future.

There were two grading runs, first we ran the grader region by region and then for the entire AS in one go. Grading the entire AS at the same time took roughly two hours.

### 5.4 Comparison with manual grades

As we had the chance to run the tester on this years mini-internet, we can compare the grader with the manual grading. Be aware that this comparison suffers from multiple issues:

- Not all questions are gradable by the grader, so we only focus on questions which are gradable.
- The points per tasks for the configuration (which we can grade) and text question which we cannot. For better comparison the manual points were reverse engineered to remove the points given for the text questions.
- The grader is quite strict, in most cases it either gives points or it doesn't.
- The grader has some issues with the rogue announcements from the stub AS. It probably has more.
- As a basis I use the databases from the region-by-region tests. One AS was missed in those tests, its database was substituted from the "entire AS in one go" test.

|                     | manual points | no manual points |
|---------------------|---------------|------------------|
| automatic points    | 143           | 4                |
| no automatic points | 55            | 14               |

Table 5.1: Compare manual grading awarding points and automatic grading awarding points

### Comparison of difference per question

To compare the manual with the automatic grading, we have taken the difference between the automatically awarded points and the manual points (adjusted to exclude points for the text questions). The differences and how often they occur are plotted in figure 5.1, with the difference on the x axis and the count on the y axis (like a histogram). A negative difference means that there were more manual points than automatic points. In most questions it is quite close to the manual grading except for question 3.1. I assume this is because the grader is very strict and this was the BGP policy question, in addition there were quite a few points for that question. In general it seems there is a tendency of the grader to be stricter than the manual grading.

#### 5.4.1 Comparison of points/no points

Due to the mostly binary nature of the grader, I find the best comparison is the following: Instead of comparing points, we compare whether points were given at all, i.e. the four scenarios are

- The grader and the manual grading gave points
- The grader did not give points, the manual grader did
- The grader gave points, manual grading did not
- The grader did not give points, neither did the manual grading

The results of this comparison are in table 5.1. Using this comparison method, automatic and manual grading agree in 157 cases or in roughly 72% of the cases. Again it seems that the grader has a tendency to be stricter than manual grading.



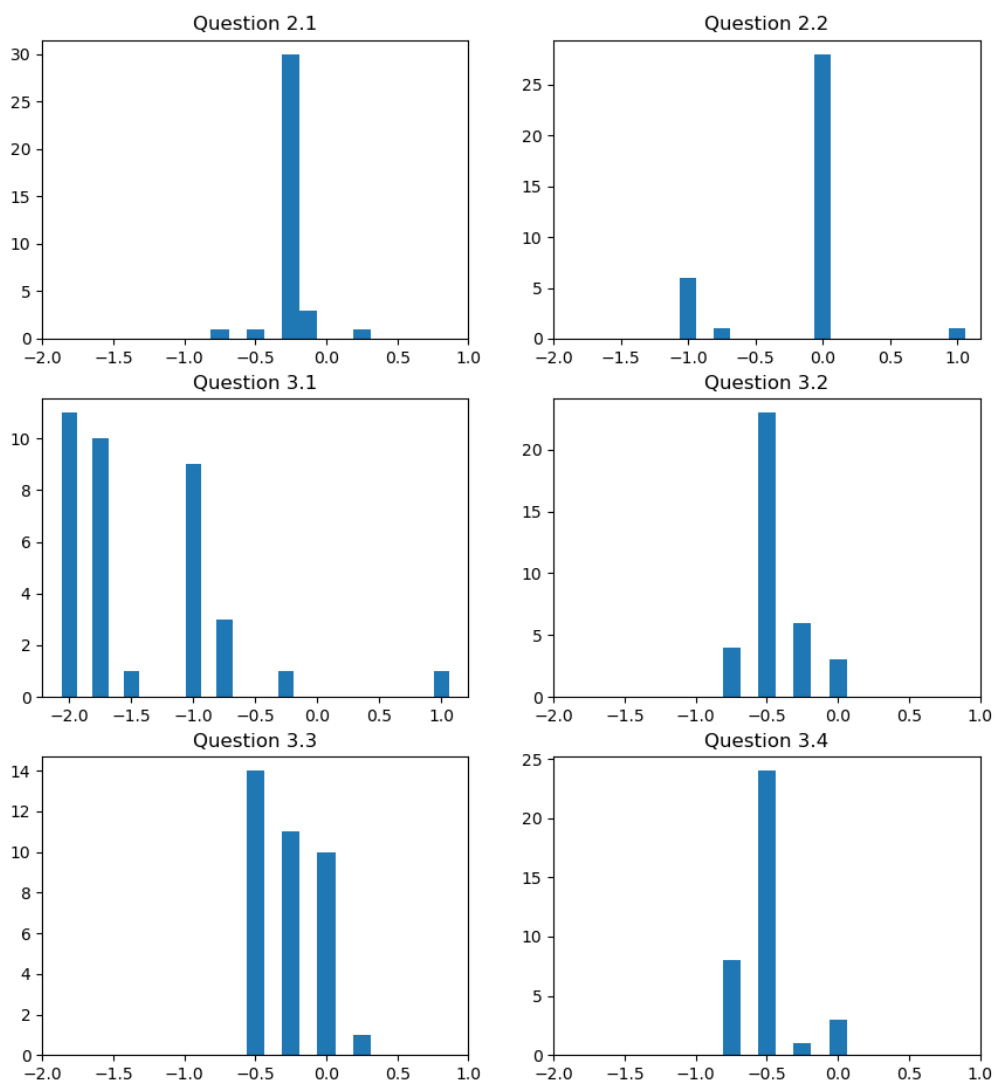


Figure 5.1: Per question difference of automatic and manual points. A negative difference means that the manual grading awarded more points, positive difference means the automatic grader awarded more points

## Chapter 6

# Outlook

Here is a short list of possible direction of future work:

- Improve connect.sh and disconnect.sh to be more generic. With that change it would be simple to also disconnect the hosts from the mini-internet and inject packets from inside the mini-internet. Another option is to add an interface to the host bridge which can sent as any host.
- Rewrite the runner in Python. This would remove the Go dependency. Also in this case the grader could make use of the databases directly. It doesn't do that because I didn't want to draw in a dependency.
- Find a way to test the L2 networks as well. As IP addresses used by the containers are assigned by students, they would have to be extracted using the ip(8) command or similar. Another way would be to have the students clearly document their chosen addresses in an easy to parse way.
- Rewrite docker exec heavy scripts to interact with the docker socket directly.
- Find out if the above works for ovs as well
- Improve the speed of getlinks.sh either with a kernel patch or by doing the ioctl's directly (depends on where the inefficient data structure is used).
- Integrating SQLite into the startup scripts of the mini-internet. There are two steps to this:
  - Have the different scripts report their actions, e.g. the ovs-docker.sh script to report the links it creates. With that the parse\_ovs.sh script could be probably made obsolete.
  - Actively use the information stored in various databases (the config database for example). This can help to simplify some of the scripts, e.g. the dns\_setup script. Below an improvement is demonstrated to a snippet:

Before:

```
# read configs
readarray groups < "${DIRECTORY}"/config/AS_config.txt
group_numbers=${#groups[@]}

# Check if there is a DNS server
is_dns=0
```

```

for ((k=0;k<group_numbers;k++)); do
    group_k=${groups[$k]}
    group_as=${group_k[1]}
    group_router_config=${group_k[3]}

    if [ "${group_as}" != "IXP" ]; then
        if grep -Fq "DNS" "${DIRECTORY}"/config/${group_router_config}; then
            is_dns=1
            break;
        fi
    fi
done
After:
# Check if there is a DNS server
is_dns=$(sqlite3 -readonly config.db << EOF
    SELECT COUNT(*)
    FROM as_config
    JOIN router_config
    ON as_config.router_config = router_config.name
    WHERE ext = 'DNS'
EOF
)

```

The variant with SQLite3 is faster (real time measured with the shell built-in time, 1000 samples each, significant difference at 99.5% confidence as reported by minostat(1)).

An improvement would be that the state can be directly read from the database and does not require reverse-engineering with ovs-vsctl(8) and ip(8). Another is that some queries can be improved (e.g. instead of iterating over all ASes and routers in dns\_setup.sh, simply let the database only return the relevant ASes and routers directly).

- Consistently name the database columns. In some places the schemas might also be improved by naming foreign keys like the column they reference in order to allow easy JOINS with USING.
- Writing or finding a proper parser for FRR configurations is probably the next step which make more complex configuration analysis possible. At the moment it is all quite hacked together and probably not very robust. Having a tool being aware of the syntax and keywords will improve that.
- With the new capabilities some new tasks might be possible for the mini-internet.
- At the moment we do not expect students to drop odiously incorrect AS paths i.e. routes coming from AS 4 with path e.g. "5 ...". In the future the grader should allow for these announcements to be blocked without deducting points.
- The looking glass checker might need some code to handle non /8 prefixes.
- Running lgparse.py with the real mini-internet looking glasses takes a few minutes, some analysis should be made to see if indexes or caching can improve the performance.

## Chapter 7

# Summary

The results of this thesis is a set of tools, which can be used to verify correct configuration of the mini-internet, collect configuration information, provide statistics about the configuration and create feedback for the students and lecturers. These tools work either passively by looking published files or actively by disconnecting ASes and interacting directly with them. The grader is in a state which allows easy expansion over the years.