Automating tests

with Anritsu traffic generators/analyzers

By: Sean Rijs

Supervisor: Ariën Vijn

January 18, 2013
Final version
Preface

This research report is part of my Bachelors of ICT graduation project at the Hogeschool van Amsterdam (HvA) and the Amsterdam Internet Exchange (AMS-IX). It describes the project I conducted to prove that I reached the level of a Bachelor of ICT degree.

I would like to thank Ariën Vijn, Stefan Plug, Michiel Appelman, Martin Pels, Niels Bakker, Eric Nghia Nguyen-Duy, includig all other employees of AMS-IX, Viki Navratilova, Arnim Eijkhoudt and Romeo Zwart for their support with this project.
Abstract

The Amsterdam Internet Exchange (AMS-IX) is an organization that offers a network platform where network organizations can exchange Internet traffic between each other. These network organizations are Internet Service Providers (ISPs), content providers and other telecommunication companies. AMS-IX strives to offer a high availability network platform.

In order to support this goal AMS-IX conducts testing procedures before implementing new software or hardware on its production platform. In these testing procedures traffic speeds up to 100Gbit/s need to be generated. General purpose computers can not generate such streams because the traffic is limited by the General Purpose Processor (GPP). For this reason AMS-IX utilizes specialized network generators manufactured by Anritsu, Japan.

These network generators are manually controlled via a Graphical User Interface (GUI). This makes executing multiple tests labor intensive. Past experiences at AMS-IX have shown that some problems in software and hardware remain unnoticed in testing procedures due to insufficient testing. With the consequence that the availability of the platform is at risk.

The traffic generators made by Anriatsu provide a remote control feature. This enables the possibility to automate the traffic generator. No existing testing system was found that supported this feature. However we found a testing system called Scapytain that partially fits the solution. Scapytain was designed for automating Scapy tests, Scapy is a traffic generator software library built for the Python programming language. Because it only runs on general purpose computers which are not sufficient for AMS-IX’s testing procedures.

An new software library for Python, named python-anritsu, was developed during this project that controls the Anritsu via its remote control feature. Scapytain is able to manage tests and python-anritsu is able to configure the Anritsu traffic generator. This system helps automating the testing procedures of AMS-IX, indirectly improving the stability of the platform.
## Contents

Abstract ................................................................. 5

1 Introduction ......................................................... 9
1.1 Problem definition ................................................. 9
1.2 Functional requirements ......................................... 9
1.3 Deliverables ....................................................... 10
1.4 Scope ............................................................... 10
1.5 Planning .......................................................... 11

2 System for automated tests ........................................... 13
2.1 Orientation ........................................................ 13
2.2 Test manager ....................................................... 15
   2.2.1 Security ..................................................... 16
2.3 Anritsu remote control library .................................. 16
   2.3.1 Anritsu remote control protocol .......................... 16
   2.3.2 Structure ................................................... 17
   2.3.3 Development process ...................................... 19
2.4 System extensibility .............................................. 20

3 Conclusion ............................................................ 21

4 Reflection ............................................................. 23

Appendices

A AMS-IX ................................................................. 25
   A.1 Internet Exchange Point ....................................... 25
   A.2 Organization .................................................... 26
   A.3 History .......................................................... 28
   A.4 Peering platform ............................................... 28
   A.4.1 High Availability .......................................... 29
   A.4.2 Hardware .................................................... 30
# AMS-IX Test Processes

B.1 Test Process - Brocade

B.2 Test Process - Hardware

# Example Python-anritsu code

# Scapyproxy

# Example Unit Test

# Python-anritsu documentation

F.1 analyzer API

F.2 port API

F.3 stream API

F.4 input_validator API

F.5 iptothex API

F.6 mactohex API

F.7 error API

# Acronyms

# References
Introduction

Chapter 1

This document represents a project commissioned by the Amsterdam Internet Exchange (AMS-IX). AMS-IX offers a neutral network platform where its members and customers can exchange traffic between each other. These members and customers are Internet Service Providers (ISPs), content providers and other telecommunication companies. AMS-IX currently connects over 540 networks. The AMS-IX organization is not directly relevant for this project. However Appendix A describes AMS-IX in more detail.

1.1 Problem definition

One of the primary goals of AMS-IX is to offer high availability for its customers.\[1\] For example, if a device on the platform fails, a backup device almost instantly kicks in. Details of this high availability design are described in Appendix A.4.1.

The software that runs on the devices needs to be regularly updated to fix (non-)critical bugs or to add features. Unfortunately there is always a possibility that new bug fixes cause regressions\[2\] to appear because of a software update.

AMS-IX uses a single vendor for its platform as described in Appendix A.4.2. Any problems such as a software bug may effect the stability of the services AMS-IX provides. To reduce this risk AMS-IX tests new hardware and software in a testing environment as described in Appendix B.

In such a testing process AMS-IX uses specialized traffic generators made by Anritsu. Configuring traffic on an Anritsu traffic generator is manually done via a Graphical User Interface (GUI). This makes executing multiple test cases labor intensive.

Past experience at AMS-IX showed that some bugs remain unnoticed due to insufficient testing. Therefore AMS-IX is searching for ways to improve its testing process. The project described in this document is part of that search.

1.2 Functional requirements

To further alleviate the risks for the stability of the platform, a system needs to be created that can run previously created tests. This prevents the problem of reconfiguring the same tests for every software update or hardware change.

---

\[1\] AMS-IX is an association. Hence it has members.

\[2\] When a bug fix is introduced, this may cause other bugs to appear - this is called regression.
The functional requirements of the system are:

- Save configured tests
- Automatically execute a defined set of tests and report the results
- Tests must use the Anritsu MD1230B/MD1260A traffic generators/analyzers

1.3 Deliverables

AMS-IX requires a system which uses the Anritsu to generate test data and automates the testing procedure using the Anritsu generated test data. There are currently no systems available that meet all of these requirements. However there is one system available which manages tests, and its tests are written as Python code. This system can be extended to also include the Anritsu. The primary deliverable of this project is to create the extended testing system, including its documentation.

The Hogeschool van Amsterdam (HvA) and AMS-IX require a thesis that supports and describes the project. This thesis is the second deliverable.

- Testing system deliverables:
  - Ability to manage tests
  - Library that controls the Anritsu
  - Documentation of the library
- Thesis

1.4 Scope

The project contains two deliverables, the system with a library and the thesis. The library is bound to the following scope:

- The development priorities are to deliver readable code, to maximize its lifetime and extensibility. Creating specific tests against bugs is excluded in this project, but some are created for code testing purposes.
- The tests must be executed via the Anritsu traffic generators/analyzers and the results must be shown on the system.
- Developing a library that automatically configures routers for a test case is excluded.
1.5 Planning

The project started at 3-9-2012 and ended on 1-2-2013. Project planning on a weekly basis is defined in Table 1.1.

<table>
<thead>
<tr>
<th>Week</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Orientation phase</strong></td>
<td></td>
</tr>
<tr>
<td>36-37</td>
<td>Learn and document what AMS-IX is about</td>
</tr>
<tr>
<td>38</td>
<td>Preliminary examination of the project</td>
</tr>
<tr>
<td>39</td>
<td>Advise concept solution</td>
</tr>
<tr>
<td>40</td>
<td>Prepare for development</td>
</tr>
<tr>
<td>First examiner visit</td>
<td></td>
</tr>
<tr>
<td><strong>Main phase</strong></td>
<td></td>
</tr>
<tr>
<td>41-44</td>
<td>Work on first proof of concept of system</td>
</tr>
<tr>
<td>45</td>
<td>Present proof of concept to AMS-IX technical departments</td>
</tr>
<tr>
<td>46-49</td>
<td>Process feedback from presentation</td>
</tr>
<tr>
<td>Development library</td>
<td></td>
</tr>
<tr>
<td>Write thesis</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>Thesis will be 70% finished</td>
</tr>
<tr>
<td>Examiner visits for a go/no-go verdict</td>
<td></td>
</tr>
<tr>
<td><strong>Finalizing phase</strong></td>
<td></td>
</tr>
<tr>
<td>51-2</td>
<td>Finish thesis</td>
</tr>
<tr>
<td>Finish the system via testing and debugging</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Thesis will be 100% finished and delivered to the graduation committee</td>
</tr>
<tr>
<td>4</td>
<td>Prepare presentation</td>
</tr>
<tr>
<td>5</td>
<td>Presentation to the graduation committee</td>
</tr>
</tbody>
</table>

Table 1.1: General planning
2 | System for automated tests

This chapter describes how the system for automated tests is designed and what changes were required to make the system complete. The reader of this chapter should have a basic understanding of software development and networking subjects like Unified Modeling Language (UML), Object-Oriented Programming (OOP) and Transmission Control Protocol (TCP).

2.1 Orientation

In the orientation phase of this project, we searched for a system that met the functional requirements. Only one system was found within the given time frame, which was Scapytain.[2]

Scapytain is related to a project called Scapy.[3] Scapy is a packet generator library for the Python programming language.[4] It allows control over all header and data fields of any protocol that it supports. For example Scapy can generate any ARP, IPv4 and IPv6 packet defined by the user, as well as various application layer protocols.

Scapy is fully implemented in software which means that it processes packets with a General Purpose Processor (GPP). This is a flexible approach, however a GPP can not generate full line-rate 10 Gigabit Ethernet (10GE) and 100 Gigabit Ethernet (100GE) traffic.[5] For this reason AMS-IX uses specialized 10GE and 100GE traffic generators. However Scapytain can be used as the test manager, provided that Scapytain additionally controls the Anritsu.

The reason why a GPP can not generate full line-rate 10GE and 100GE, is because a GPP has to perform generic operating system tasks. Tests show that a GPP can not achieve 10GE and certainly not 100GE speeds.[5] However a specialized traffic generator can achieve these speeds, because it moves the network processing task to a Field-Programmable Gate Array (FPGA). An FPGA is specifically programmed for network processing, and can handle the traffic intensity 10GE and 100GE requires.[5, 6] With specialized network generators the GPP controls the FPGA and configures it to send or receive traffic. Figure 2.1b describes the architecture of a FPGA network generator.
Chapter 2

(a) Processing traffic with a GPP.
(b) Processing traffic with an FPGA.

Figure 2.1: Comparison of traffic processing between GPP and FPGA.
2.2 Test manager

Scapytain is a test manager that runs as a web application which executes, saves, and stores test reports in a database. Tests are written as Python code. Scapytain is organized into tests, test plans, test means, and campaigns, as depicted in Figure 2.2. Tests are simply a few lines of Python code, which should be programmed to return True if a test passes or False when it does not. Its original use case is to use Scapy functions as input of these tests.

Because of this behavior an additional Python library can be developed that controls the Anritsu generator. The test code in Scapytain imports this library, consequently enabling Scapytain to create tests using an Anritsu. To understand the structure of such a system combining Scapytain and a Python library, see the UML deployment diagram in Figure 2.3 on page 16.
2.2.1 Security

Scapytain always imports the Scapy library during execution. Scapy does not use the operating system sockets to send and receive packets, instead it directly communicates with the network card. Because of this behavior it requires root privileges.

The user that runs Scapytain is the same user that runs the web application. However running a web application under the user root is not considered a best practice. To counter this problem Scapytain implemented a proxy feature called scapyproxy, where Scapy can be remotely imported and run it via a proxy using for example Secure Shell (SSH). With this feature Scapytain can be run as an unprivileged user, while running a privileged Scapy process on a different host.

AMS-IX does not require Scapy for now, and if the need for Scapy is ever required it can be easily configured. Appendix D describes the required steps to configure scapyproxy. In the current implementation Scapy is run with user privileges, making it unable to generate packets.

2.3 Anritsu remote control library

I have named the software library that controls the Anritsu python-anritsu because it is designed to use with Python and to control an Anritsu. The python-anritsu library can be briefly described as an abstraction layer on top of the Anritsu remote control protocol. This section describes the remote control protocol and the library in more detail.

2.3.1 Anritsu remote control protocol

The Anritsu enables remote control over TCP or a General Purpose Interface Bus (GPIB) bus. The GPIB bus is a standard specified by the Institute of Electrical and Electronics Engineers (IEEE) as IEEE-488. This standard is divided into two parts, IEEE-488.1[7]
IEEE-488.1 defines the hardware (i.e., the GPIB bus) and IEEE-488.2 defines the protocol.

Python-anritsu uses TCP instead of the GPIB bus. The reason for this choice is because GPIB requires users to attach a cable to the web server, which is not practical if the web server is not physically close to the Anritsu. With TCP the web server can stay at the same physical location as long as it can reach the Anritsu over TCP.

The Anritsu can be controlled with two types of messages, as defined by IEEE-488.2. These are command messages which set a variable value, and query messages which return the variable value. Generating packets is done by configuring a Stream, for example a stream can represent: send 10.000 Ethernet frames with Source Address f4-6d-04-36-c4-2e and destination address 68-05-ca-0f-88-70. Every port has its own Stream Table in which to configure streams. Figure 2.4 on page 18 describes this structure of the Anritsu.

The following line is a command message to configure a stream on generator 1, module 5, port 1 with Ethernet Source Address f4-6d-04-36-c4-2e:

```
:UENTry:ID 1
:MODule:ID 5
:PORT:ID 1
:TSTReam:TABLE:ID 1
:TSTReam:TABLE:ITEM:FRAMe:ETHernet:SA:VALue #HF46D0436C42E
```

However nothing will be returned from the Anritsu. We have to create a new message - a query - to test if the variable is set with our requested value. A query is similar to a command, except it ends with a question mark instead of a value. When received the Anritsu returns the requested variable value. The following line is a query to test if our original variable is set:

```
:TSTReam:TABLE:ITEM:FRAMe:ETHernet:SA:VALue?
```

Then the Anritsu returns the variable value:

```
#HF46D0436C42E
```

A query must also be used to get a counter of some sort. For example, if we want to know the number of received frames on unit 1, module 5, port 2, the command and the query should be respectively:

```
:UENTry:ID 1
:MODule:ID 5
:PORT:ID 1
:COUNter:TRANsmitted:FRAMes?
```

Then the Anritsu returns the variable value:

```
10000, 0
```

The first number represents the received packets and the second number the received packets per second.

### 2.3.2 Structure

This section describes the structure of python-anritsu, and explain the decisions made during its development.
System for automated tests

Chapter 2

Figure 2.4: Anritsu structure

From a functional perspective python-anritsu’s purpose is to translate the IEEE-488.2 protocol to Python functions. The user is not required to learn the IEEE-488.2 protocol in order to use the library. See Appendix C for example code of how a user can interact with python-anritsu.

The library has the following technical requirements:

- Not all stream types can be run on all Anritsu device types, therefore streams must be aware on which analyzer type the stream is being configured for.

- The IEEE-488.2 protocol does not return the success or failure of a command. The library must automatically test if the user input is processed on the Anritsu.

- The Anritsu can have multiple streams on multiple ports. Python-anritsu must be flexible enough to control multiple Anritsu analyzers, while an analyzer can control multiple ports, and a port can have multiple streams.

Python-anritsu is designed to fit these requirements using OOP. The UML class diagram depicted in Figure 2.5 on page 20 describes this design. When the analyzer class is assigned to a variable, the derived instance\(^1\) represents a single Anritsu analyzer. When the user wants to create a stream, it calls a method via this analyzer instance, which returns an instance representing a stream. When a stream is created for the first time, the analyzer instance can pass its variables to the stream instance, making it aware which type of Anritsu the stream will be configured on. This completes the first requirement.

When a stream is configured, it must be committed via the analyzer instance. When committing a stream the commands\(^2\) are first sent. This is followed by a check that sends query messages to check if the Anritsu variables values requested are actually set on the Anritsu. If not, it will raise an error with a description, consequently failing the test. To start or stop transmitting a stream, the analyzer instance is used in the same way. Because of this OOP structure multiple analyzers can be controlled. This completes the second and last requirement.

\(^1\)Also known as an object

\(^2\)The definition of a command and a query is described in subsection 2.3.1
2.3.3 Development process

There exist many development methods for developing software. AMS-IX does not follow any of these development methods, because from AMS-IX’s perspective there was never a need for it. However some parts of development methods can be found in this project. This subsection describes these parts in the development of the python-anritsu library.

Unit Tests

The Stream class as depicted in Figure 2.5 on page 20 can grow up to about 100 or more functions, assuming all the traffic stream properties are implemented. Note that if code is changed, the behavior of the Stream class at run-time is affected, even to the extent that it becomes broken. To alleviate this problem the functions need to be tested. Unfortunately manual testing after each change is rather time consuming.

The solution to this problem is to create a script that automatically executes the Stream class and tests if functions still behave as expected after a change. Such scripts are called unit tests. A unit test is just a class containing functions that tests only one specific part of a system. It is possible to extend unit testing to other classes in python-anritsu. However these classes are rather small and the changes expected are minimal. Therefore the decision was made to not to spend time in developing a unit test for other classes.

Appendix E shows the unit tests for the Stream class.

Documentation

In order to increase the life time and extensibility of python-anritsu it is a high priority goal to make the code understandable to other developers. An important part of reaching this goal is the documentation of the code. This means the code needs comments that describe what classes, functions and code snippets are doing and for what purpose. Additionally, it helps to document the code outside the source code. But doing the same work twice is inefficient.

A documentation tool called Sphinx alleviates this problem. This tool extracts docstrings from Python source code converting it to formatted HyperText Markup Language (HTML) pages. This prevents the developer from having to write the same documentation in different places. The contents of the python-anritsu documentation is imported to Appendix F.

Code Style

The Python code is written following the guidelines described in Python Enhancement Proposal (PEP)-8. PEP-8 is one of the early proposals in the Python community which gives coding conventions for writing Python code. Its goal is to improve readability by giving guidelines in what style to write code. For example, classes should be written using the CapsWords convention and, functions should be in lowercase while separating words with underscores.

In Python, comments for modules, classes and functions are called docstrings
Chapter 2

2.4 System extensibility

In the final phase of the project, the system using Scapytain and python-anritsu was tested with the testing process described in Appendix B.2. This test was done to confirm that the system can be used in a practical situation.

After the test was done, AMS-IX noted that it still required repetitive work to configure a router for a given test case. Another library like python-anritsu could be written to automatically configure a router for a given test case. To address this requirement, a subproject was started to develop such a library. When this subproject delivered its first proof of concept, it was imported into Scapytain in the same way python-anritsu is.

Extending the system with more libraries is easy. The new libraries have only the following technical requirements:

- They must be run non-interactively.
- A function that succeeds must return zero and when it fails it must return one or higher.

![Analyzer UML Class Diagram]

![Stream UML Class Diagram]

Figure 2.5: UML Class Diagram
3 Conclusion

In the introduction chapter we defined the problem and what is required to solve that problem. The list of functional requirements were:

- Save configured tests
- Automatically execute a defined set of tests and report the results
- Tests must use the Anritsu MD1230B/MD1260A traffic generators/analyzers

The first and second requirements were satisfied by using the test manager Scapytain. The last requirement was satisfied by the python-anritsu library which is able to integrate into Scapytain. Altogether it created a testing system which met all the requirements.

To test if the system worked in a more practical situation, the testing process of hardware was replicated as described in Appendix B.2.

The hardware test results were positive, however AMS-IX wanted to further extend its functionality by automatically configuring a router. A subproject was created to develop a new library like python-anritsu which included this functionality. This subproject proved that the system is easy to extend with more functionality.
4 | Reflection

Before I was allowed to start this project, the HvA graduation committee asked me a few questions. One of the questions was what competence points for a Bachelor degree will be met in this project. This chapter reflects on those initial questions.

Table 4.1 describes the competence points I initially expected to execute in this project, followed by subsections that describe how I have shown these type of competences. The competences are taken from the HvA study guide[11], and translated to English.

<table>
<thead>
<tr>
<th>Competence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Plan</strong></td>
<td><strong>Research &amp; design</strong></td>
</tr>
<tr>
<td>P1</td>
<td>Map out the (information)requirements of different disciplines</td>
</tr>
<tr>
<td>P2</td>
<td>Translate the acquired (information) requirements to a functional design</td>
</tr>
<tr>
<td>P3</td>
<td>Elaborate a functional design and advise the client</td>
</tr>
<tr>
<td><strong>Evaluate</strong></td>
<td><strong>Evaluate &amp; advise</strong></td>
</tr>
<tr>
<td>E2</td>
<td>Determine the consequences for the users of change in an existing system</td>
</tr>
<tr>
<td><strong>Task</strong></td>
<td><strong>Task oriented</strong></td>
</tr>
<tr>
<td>T1</td>
<td>Systematic approach: the student is able to systematically apply its tasks</td>
</tr>
<tr>
<td>T2</td>
<td>Project structure: the student is able to work in a project structure</td>
</tr>
<tr>
<td>T3</td>
<td>Assignment testing: the student is able to handle their own responsibilities</td>
</tr>
<tr>
<td><strong>Relationship</strong></td>
<td><strong>Relationship oriented</strong></td>
</tr>
<tr>
<td>R1</td>
<td>Communication: the student is able to communicate in a spoken and written manner</td>
</tr>
<tr>
<td>R2</td>
<td>Collaboration: the student can collaborate with others</td>
</tr>
<tr>
<td>R3</td>
<td>Empathy: the employee is able to work with people that have a different view of the world</td>
</tr>
<tr>
<td>R4</td>
<td>Relationship management: the employee is able to maintain relations with others</td>
</tr>
<tr>
<td><strong>Self</strong></td>
<td><strong>Self oriented</strong></td>
</tr>
<tr>
<td>Z1</td>
<td>Reflection: the employee is able to adjust and measure their own effect and efficiency in his or her tasks</td>
</tr>
<tr>
<td>Z2</td>
<td>Learn: the student is able to learn and apply new knowledge</td>
</tr>
</tbody>
</table>

Table 4.1: Table of competence for Bachelor of Information and Communication Technology
Plan: P1, P2, and P3

The technical departments’ are more or less experienced in the same discipline, because the corresponding employees have an indepth knowledge about network technology. However the design priorities can differ, as the Network Operations Center (NOC) employees asked for more changes in the usability of the system, while the engineering and security employees ask for stronger security in the system. These requirements were discussed together and a consensus was formed which created the design described in subsection 2.2.1.

Evaluate: E2

The primary change in the existing testing processes is that Anritsu configurations are scripted in Python instead of manually configuring on the GUI of the Anritsu. This can automate a list of test cases. The secondary change is that users have to visit a web application to manage the tests.

Task: T1, T2, and T3

A plan was created to outline the general deadlines as described in Section 1.5. When a task was done it was demonstrated to the supervisor. If there was any feedback it was applied back to the task. There was no specific plan for each task, because there was no need for it as described in Section 2.3.3. Despite the lack of specific planning however, the project still followed a systematic and responsible approach, shown by the results described in Chapter 3.

Relationship: R1, R2, R3, and R4

This project required frequent communication. When a task was done it was demonstrated, this was usually done to the supervisor of this project in a spoken manner. When this was not possible a written communication manner like E-mail or Instant Messaging (IM) was used. To demonstrate the current status of the project, two internal presentations were given to the technical departments of AMS-IX. The system was also tested together with an employee of the NOC department to gain feedback on the system.

Self: Z1 and Z2

In order to be more efficient and effective in development, I learned to write unit tests and generate automatic documentation as described in Section 2.3.3. This increased the stability of my code and reduced the time spent in writing documentation, which consequently increased my efficiency.

\[\text{The AMS-IX technical departments are described in Appendix A.2}\]
In this appendix AMS-IX is described. It begins by first explaining what an Internet Exchange Point (IXP) is. Little to no technical knowledge is required to understand this section, as long as the reader understands that the Internet is simply made of networks connected to each other. The second section describes the history of AMS-IX. This shows how fast AMS-IX has grown in the past 18 years. Then I describe how the organization is structured. The last section is about the peering platform where all customers connect. This requires technical knowledge about Transmission Control Protocol/Internet Protocol (TCP/IP) in general, the Open Systems Interconnection (OSI) model, routing protocols Border Gateway Protocol (BGP) and the Multi Protocol Label Switching (MPLS)/Virtual Private LAN Service (VPLS) mechanism. It is also assumed the reader knows what Virtual Local Area Network (VLAN) and Simple Network Management Protocol (SNMP) are.

A.1 Internet Exchange Point

AMS-IX is an IXP. An IXP offers a platform where organizations with a relatively great deal of network traffic, such as ISPs and content providers, connect to each other.

To understand the position of AMS-IX in the structure of Internet, the tier hierarchy of the Internet must be understood. The tier hierarchy is a theoretical model of how the Internet is structured. Figure A.1a on page 26 shows how the networks in the tier hierarchy are structured. Tier 1 has indirect access to the whole Internet. If a tier 2 or 3 provider wants to exchange traffic with a network which is not directly connected, then it needs to send its traffic upstream to a network with a lower tier number. From the perspective of the tier 2 and 3 networks the upstream networks are called transits. Transits usually bill on a per bit basis. Additionally it takes more hops to reach a destination via a transit, because the transit is typically not as close to the edge as lower tier numbers. In general this means a transit costs more and needs more hops in contrast to a situation where traffic goes downstream to a directly connected network.

From a cost perspective, networks ideally never want to exchange traffic with their transits and only exchange traffic downstream, to a lower or equal tier. This is where IXPs offers a solution. Figure A.1b on page 26 shows where an IXP like the AMS-IX is positioned in the tier hierarchy. The networks connect to each other via AMS-IX. This connecting is called peering. IXPs generally bill flat fees in contrast to transits. This results in less hops and lower costs, assuming the network exchanges some minimal amount of traffic to the IXP.

Most of the connected networks still use a transit in case they don’t have access to a network via their IXP, or as a backup if there are problems with their connection to the IXP.
AMS-IX consists of the AMS-IX Association and AMS-IX B.V. The AMS-IX Association is the only shareholder of AMS-IX B.V. The organizations that connect to AMS-IX can become a member of the AMS-IX Association and are invited to attend certain meetings. A management overview of the organization is displayed in Figure A.2 on page 27.

The organization’s Chief Executive Officer (CEO) is Job Witteman, who leads the organization and gives structure to it.

The Chief Marketing Officer (CMO) is Cara Mascini, who is responsible for staying within the company’s philosophy when working to increase sales growth and promotional goals.

The Chief Financial Officer (CFO) is Marco Bakker, who is responsible for Human Resources and Financial Administration. The goal for a financial administrator is insight into the financial status of AMS-IX at any time.

The European Internet Exchange Association (Euro-IX) secretariat is Bijal Sanghani, who runs and administers the secretariat services for the Euro-IX.
at AMS-IX for Euro-IX but is not part of the AMS-IX management team. Euro-IX defines itself in the following way: “Euro-IX is an association of Internet Exchanges, promoting an open interchange of ideas and experiences, gained to mutual advantage of the membership, by offering fora, meetings, mailing lists and on-line resources. Euro-IX also gathers information on regulatory issues affecting member exchanges within the region and where appropriate from other jurisdictions that could potentially impact on the membership.”

The Chief Technology Officer (CTO) is Henk Steenman, who manages the technical staff. The staff can be divided into Supporting, Operations, Engineering and Security branches. The Supporting branch, called Sysops, and Development, is managed by Steven Bakker. He is responsible for the organization’s internal network, including systems and website management and development. The Operations branch is named NOC and is managed by Martin Pels. The NOC is responsible for managing and developing the peering platform, which is the core business of the organization. The Security branch is managed by Niels Bakker, who is the Security Officer of the organization. The Engineering branch is managed by Ariën Vijn. He manages the network for production. Which in turn gets delivered to the NOC. The Engineering branch is where I am positioned as an intern.

AMS-IX currently has 38 employees. 52% of the employees have a Dutch background and the remaining have a foreign background. The spoken and written language in AMS-IX is therefore English.

![AMS-IX organizational chart](image-url)
A.3 History

In 1994 a platform was created where academic institutes with the European Organization for Nuclear Research (CERN) were connected, with ISPs were also allowed to connect to this platform. An IXP was thus created and named AMS-IX. In 1997 the AMS-IX Association was founded by those connected ISPs. In 1998 the multicast VLAN was implemented and the first Internet Protocol version 6 (IPv6) tests were done. In 1997 the AMS-IX Association was formed. The day-to-day management was outsourced to Surfnet, and Surfnet outsourced the technical management to Stichting Academisch Rekencentrum Amsterdam (SARA). In 2000 AMS-IX B.V. a limited liability company was formed to take the day-to-day management over from Surfnet. The technical management remained at Surfnet, but in 2001 it was in-sourced as well. In 2003 AMS-IX had a total of 178 members and had a marketing and customer relations department. In 2005 this has increased to 234 members. In 2009 a sales department was added. Currently AMS-IX has 515 members.\[12\]

A.4 Peering platform

The AMS-IX peering platform is located in multiple physical locations.\[13\] The locations are listed in Table A.1. The first column shows the organization responsible for the datacenter, the second column is the physical location. Their logical positions in the network are shown in Figure A.3.

<table>
<thead>
<tr>
<th>Datacenter</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>TelecityGroup</td>
<td>AMS2 - Amsterdam Southeast</td>
</tr>
<tr>
<td></td>
<td>AMS5 - Amsterdam Southeast</td>
</tr>
<tr>
<td></td>
<td>AMS4 - Amstel Business Park (Extension to Southeast)</td>
</tr>
<tr>
<td>Global Switch</td>
<td>Amsterdam Slotervaart</td>
</tr>
<tr>
<td>Equinix</td>
<td>AMS1/2 Amsterdam Slotervaart</td>
</tr>
<tr>
<td></td>
<td>AMS3 - Amsterdam Science Park</td>
</tr>
<tr>
<td>Terremark</td>
<td>Schiphol</td>
</tr>
<tr>
<td>EvoSwitch</td>
<td>Haarlem</td>
</tr>
<tr>
<td>InterXion</td>
<td>Schiphol-Rijk</td>
</tr>
<tr>
<td>euNetworks</td>
<td>Amsterdam Amstel Business Park</td>
</tr>
<tr>
<td>Vancis (SARA)</td>
<td>Amsterdam Science Park</td>
</tr>
<tr>
<td>NIKHEF</td>
<td>Amsterdam Science Park</td>
</tr>
</tbody>
</table>

Table A.1: AMS-IX physical locations

Physically the peering platform is an extended star type of network topology, because there are multiple cores. Recently AMS-IX upgraded their platform with two sub-cores as depicted in Figure A.3. In Cisco terms this is called a distribution layer. These sub-cores were created to offload traffic from the Equinix and TelecityGroup locations to the cores. This keeps the traffic of those locations more local which increases the performance.

The peering platform is a MPLS/VPLS network. MPLS is mechanism that keeps track of every path between all the hosts within its MPLS network.\[14\] These paths are called Label Switched Paths (LSP). Logically the peering platform is a full mesh type of

---

1Surfnet, NLnet, AT&T EMEA, Unisource, BT, KPN Qwest, XS4All, Global One, Euronet, EUnet, Wirehub, Belnet, RIPE NCC, Demon, IXE/PSI, Telecom Finland, IBM GN, A2000, UUnet/MCI, GTS Europe (Ebone)
network topology, because each host is reached over a different LSP. MPLS paths can be automatically created using the Label Distribution Protocol (LDP). Because LDP creates automatically created paths, less control of the paths is possible. Therefore AMS-IX maintains their paths automatically via their self written LSP management tool called \textit{lspctl}.

VPLS provides customers the ability to share their local layer 2 network over an MPLS network, thus creating a VLAN. The VPLS paths are exchanged via LDP. The customers tag their frames for this VLAN with a registered VLAN number. AMS-IX routes it to the ports which are members of this VLAN.

Figure A.3: AMS-IX locations overview

Figure A.4 on page 31 shows the paths on a typical peering connection on the AMS-IX platform. The top two circles are routers from a customer, named Customer Edge (CE). The CE devices connect to a Glimmerglass 100 or 300 Photonic Cross-Connect (PXC) system. The PXC connects to the Provider Edge (PE) routers. The provider here is AMS-IX. After the paths go through the PE, the paths continue through the core and to other customers to enable connectivity with them.

The described network structure is not visible from the customer’s perspective. It is completely transparent to customers. The customers peer with each other as if they are directly connected to a single layer 2 switch.

A.4.1 High Availability

The peering platform has a high availability design. The devices depicted in Figure A.5 on page 32 are now explained in more detail to describe the high availability design.

A PXC is an automated patch panel which works on a physical layer to output traffic to another port when commanded. This process is automated by a tool written by AMS-IX. This tool consists of a daemon named \textit{pxcd}, which runs on a server in the AMS-IX network. It listens to SNMP traps and client commands. When an LSP on a PE changes state, an SNMP trap is received by the daemon. In case there are little to no available LSPs, \textit{pxcd} instructs the relevant PXC to output traffic to a different port which ends at a backup PE. Figure A.5 demonstrates a scenario to illustrate what happens to the paths in this case.
The routers are redundantly set up. In the case of a hardware failure, the other router takes over. MPLS takes care of making the routers on the platform aware of the link state. In this redundant design software updates or hardware changes are made without maintenance downtime. One router gets upgraded and tested, while the traffic moves over the other router.

A.4.2 Hardware

All routers in the peering platform depicted in Figure A.4 are from a single vendor: Brocade. Except for the last device before the CE. There the vendor changes to a Glimmerglass. The networks connect with either Gigabit Ethernet (GE), 10GE, 100GE or aggregated forms of these connections. Brocade MLX-8 routers are used for GE connections and Brocade MLX-16 or MLX-32 routers are used for 10GE and 100GE connections.

A.4.3 Peering

In the IXP subsection we explained that when networks interconnect they are peering. From a technical perspective this is not that simple. To exchange routes with a peer, the source must first know which Internet Protocol version 4 (IPv4) or IPv6 network lays behind the destination peer. Networks are assigned an Autonomous System Number (ASN) which represents a network. An ASN is unique and registered with the Internet Assigned Numbers Authority (IANA), similar to an Internet Protocol (IP) address is registered.

An ASN number is configured via an exterior gateway type of routing protocol. The common protocol used on the Internet is BGP.[15] With BGP, peers can create sessions with other peers via their ASN and can apply policies to those sessions.

Additionally AMS-IX provides two route servers with which a customer can create a BGP session. If a customer wants to connect to all or a subset of peers, the customer only has to create BGP sessions to both AMS-IX route servers. This saves a lot of administrative work compared to configuring every peer individually.
Figure A.4: Simplified architecture of the AMS-IX platform
Figure A.5: Scenario: path failure to PE
B | AMS-IX Test Processes

B.1 Test Process - Brocade

A Brocade testing process checks to see if a new software update is ready to be implemented on the production platform of AMS-IX. The testing process for AMS-IX the Brocade devices is depicted in Figure B.1 on page 34. Due to the fact that the peering platform consists of Brocade devices, we use the Brocade in our examples to describe the testing process. However other devices follow similar testing processes.

There are two kinds of updates: big updates and small updates. A big software update usually consists of many new features and bug fixes. After the initial release, the update first enters a preliminary testing process called an Early Field Trail (EFT), which is held in the AMS-IX lab. The NOC department is introduced to the new software update for the first time and runs some basic tests. Then it the upgrade undergoes an activity called a Customer Acceptance Testing (CAT). For CAT the AMS-IX peering platform is replicated at a Brocade lab, which is organized by Brocade for AMS-IX. Hardware used by AMS-IX such as the Glimmerglas PXCs and the network interfaces used at AMS-IX are installed at the lab. A few AMS-IX employees travel to this lab and they work with Brocade to find bugs in the new software update. Using the configuration that is also used on the production platform. At the end of this activity, the bugs found are reported to Brocade.

Brocades these bugs and then releases a small update with this fixes. When the small update is released to fix the reported bugs, AMS-IX applies the update in its own lab. To trigger these bugs again in their own lab. If the bugs are found again, AMS-IX report the bugs to Brocade and the small update process restarts. If no bugs are found, the update passes this testing process.

If a software update passes the first testing process, the software update is allowed to continue to the final testing process on the peering platform. On the peering platform all Brocade devices are redundantly set up, so every device has a mirror as depicted in Figure A.4 on page 31. The final testing process is depicted in Figure B.2. The update gets installed on one out of the two devices. If the device stays operational after a period of time and does not show new bugs, then the other device will be upgraded as well and the testing phase is complete.
Figure B.1: The first testing process

Figure B.2: The final testing process
B.2 Test Process - Hardware

Whenever a new module with interfaces needs to be added to a router or a new device is implemented, it must first pass a testing process to check if the hardware works. Only then it is allowed to be used for production.

An example: when a 10GE module is added with 8 interfaces, an Anritsu is connected to port 1 of the module. Port 2 is connected to 3, port 4 connected to 5, port 6 connected to 7, and the last port back to the second port of the Anritsu. The port combinations are separated via untagged VLANs. This setup is depicted in Figure B.3. At AMS-IX this is called a snake. When a snake is set up, the test is started by letting the Anritsu send a amount of Ethernet frames of port one and expects the same amount of frames to be received from port 8. If the expected amount is correct, the interfaces passed the test.

![Figure B.3: A snake test](image)

Figure B.3: A snake test
```python
#!/usr/bin/python
from anritsu import *
meetnix = analyzer.Analyzer('meetnix.noc.ams-ix.net', 'md1230b')
def set_port1():
    # configuring the first Anritsu port
    stream1 = meetnix.stream('1', '1', '5', '1')
    stream1.distribution('STOP')
    stream1.burst_per_stream('1')
    stream1.frames_per_burst('1000000')
    stream1.frame_size('AUTO')
    stream1.frame_source_address('00-00')
    stream1.frame_destination_address('76-d6')
    stream1.protocol('ARP')
    stream1.arp('ARP/request', '00-DE-AA-00-00-01', '192.168.1.1', '00-DE-AA-00-00-02', '192.168.1.2')
    meetnix.stream_commit(stream1)

def set_port2():
    # configuring the second Anritsu port
    stream2 = meetnix.stream('1', '1', '5', '2')
    stream2.distribution('STOP')
    stream2.burst_per_stream('1')
    stream2.frames_per_burst('1000000')
    stream2.frame_size('AUTO')
    stream2.frame_source_address('00-00')
    stream2.frame_destination_address('27-c7')
    stream2.protocol('ARP')
    stream2.arp('ARP/request', '00-DE-AA-00-00-01', '192.168.1.1', '00-DE-AA-00-00-02', '192.168.1.2')
    meetnix.stream_commit(stream2)
def start():
    # the testing logic including when to count, start, stop, and finally compare ports with input
    meetnix.count('1', '5', '1')
    meetnix.count('1', '5', '2')
    meetnix.transmit('1', '5', '1')
    meetnix.transmit('1', '5', '2')
    meetnix.stop_all('1', '5', '1', 'STOP')
    meetnix.stop_all('1', '5', '2', 'STOP')
    a = meetnix.get_port_counter('1', '5', '1', 'rxframes', 1000000, 1000000)
    b = meetnix.get_port_counter('1', '5', '2', 'rxframes', 1000000, 1000000)
    meetnix.disconnect()
    return compare.counter(a, b)

# the main
set_port1()
set_port2()
start()
```

C | Example python-anritsu code
In order to configure Scapytain to use a Scapy proxy, edit its config file located at 
/etc/scapytainrc. Uncomment the scapyproxy variable to the following example:

```bash
scapyproxy = ssh probe@10.0.0.10 sudo scapytain_scapyproxy
```

The above example runs Scapy methods over SSH on IP address 10.0.0.10 using sudo.

If this variable remains empty or commented, Scapy is executed under the user Scapytain
is running.
import unittest2

from anritsu import *

class TestStreamFunctions(unittest2.TestCase):
    def test_init(self):
        stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
        self.commands=[':TSTReam:TABLe:ADD
', ':TSTReam:TABLe:ID/uni24231
'
        self.assertListEqual(stream1.commands, self.commands)

def test_distribution(self):
    for argument_1 in self.argument_1_set:
        if argument_1 == 'JUMP':
            stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
            argument_2 = '256'
            stream1.distribution(argument_1, argument_2)
            self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:CONTrol:DISTribution/uni2423' + argument_1 + '
')
            self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:CONTrol:JTID/uni2423' + argument_2 + '
')
            del stream1
        elif argument_1 == 'JUMP_COUNT' or argument_1 == 'JUMP_STOP':
            stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
            argument_2 = '256'
            argument_3 = '16000000'
            stream1.distribution(argument_1, argument_2, argument_3)
            self.assertEqual(stream1.commands[-3], ':TSTReam:TABLe:ITEM:CONTrol:DISTribution/uni2423' + argument_1 + '
')
            self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:CONTrol:JTID/uni2423' + argument_2 + '
')
            self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:CONTrol:COUNt/uni2423' + argument_3 + '
')
            del stream1
        else:
            stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
            stream1.distribution(argument_1)
            self.assertListEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:CONTrol:DISTribution/uni2423' + argument_1 + '
')
            del stream1

def test_inter_burst_gap(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    argument_1 = str(1200000000000)
    stream1.inter_burst_gap(argument_1)
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:CONTrol:GAP:IBG/uni2423' + argument_1 + '
')

def test_inter_stream_gap(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    argument_1 = str(1200000000000)
    stream1.inter_stream_gap(argument_1)
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:CONTrol:GAP:ISG/uni2423' + argument_1 + '
')

def test_inter_frame_gap(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    argument_1 = str(1200000000000)
    stream1.inter_frame_gap(argument_1)
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:CONTrol:GAP:IFG/uni2423' + argument_1 + '
')

for argument_1 in self.argument_1_set:
stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
if argument_1 == 'FIXED':
    argument_2 = str(1200000000000)
    stream1.inter_frame_gap(argument_1, argument_2)
    self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:CONTrol:GAP:IFG:TYPE/uni2423' + argument_1 + '
')
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:CONTrol:GAP:IFG:VALue/uni2423' + argument_2 + '
')
    del stream1
elif argument_1 == 'RANDOM':
    argument_2 = str(96)
    argument_3 = str(1200000000000)
    stream1.inter_frame_gap(argument_1, argument_2, argument_3)
    self.assertEqual(stream1.commands[-3], ':TSTReam:TABLe:ITEM:CONTrol:GAP:IFG:TYPE/uni2423' + argument_1 + '
')
    self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:CONTrol:GAP:MINimum/uni2423' + argument_2 + '
')
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:CONTrol:GAP:MAXimum/uni2423' + argument_3 + '
')
    del stream1
def test_burst_per_stream(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    argument_1 = str(1099511627775)
    stream1.burst_per_stream(argument_1)
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:CONTrol:BPSTream/uni2423' + argument_1 + '
')
def test_frames_per_burst(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    argument_1 = str(1099511627775)
    stream1.frames_per_burst(argument_1)
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:CONTrol:FPBurst/uni2423' + argument_1 + '
')
def test_frame_size(self):
    self.argument_1_set = ['AUTO', 'FIXED', 'INCREMENT', 'RANDOM']
    for argument_1 in self.argument_1_set:
        stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
        if argument_1 == 'AUTO':
            stream1.frame_size(argument_1)
            self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:FSIZe:TYPE/uni2423' + argument_1 + '
')
            del stream1
        elif argument_1 == 'FIXED':
            argument_2 = str(64)
            stream1.frame_size(argument_1, argument_2)
            self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:FSIZe:TYPE/uni2423' + argument_1 + '
')
            self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:FSIZe:VALue/uni2423' + argument_2 + '
')
            del stream1
        elif argument_1 == 'INCREMENT':
            argument_2 = str(64)
            argument_3 = str(1518)
            stream1.frame_size(argument_1, argument_2, argument_3)
            self.assertEqual(stream1.commands[-3], ':TSTReam:TABLe:ITEM:FSIZe:TYPE/uni2423' + argument_1 + '
')
            self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:FSIZe:MINimum/uni2423' + argument_2 + '
')
            self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:FSIZe:MAXimum/uni2423' + argument_3 + '
')
            del stream1
        elif argument_1 == 'RANDOM':
            argument_2 = str(64)
            argument_3 = str(1518)
            stream1.frame_size(argument_1, argument_2, argument_3)
            self.assertEqual(stream1.commands[-3], ':TSTReam:TABLe:ITEM:FSIZe:TYPE/uni2423' + argument_1 + '
')
            self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:FSIZe:MINimum/uni2423' + argument_2 + '
')
            self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:FSIZe:MAXimum/uni2423' + argument_3 + '
')
            del stream1

def test_frame_source_address(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    stream1.frame_source_address('00-26-9e-58-27-c7', '00-00-00-ff-ff-ff', 'RANDOM')
    self.assertEqual(stream1.commands[-3], ':TSTReam:TABLe:ITEM:FRAMe:ETHernet:SA:VALue/uni2423#H00269E5827C7
')
    self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:FRAMe:ETHernet:SA:MASK/uni2423#H00000000FFFF
')
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:FRAMe:ETHernet:SA:VALUE/uni2423#H00269E5827C7
')
Example Unit Test

```python
self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:FRAMe:ETHernet:SA:TYPE/uni2423RANDOM
')
def test_frame_destination_address(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    stream1.frame_destination_address('00-24-D6-0b-76-D6', '00-00-00-FF-FF-FF', 'RANDOM')
    self.assertEqual(stream1.commands[-3], ':TSTReam:TABLe:ITEM:FRAMe:ETHernet:DA:VALue/#H0024D60B76D6
')
    self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:FRAMe:ETHernet:DA:MASK/#H000000FFFFFF
')
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:FRAMe:ETHernet:DA:TYPE/uni2423RANDOM
')

def test_arp(self):
    stream1 = stream.Stream('1', '1', '2', '3', 'md1230b')
    stream1.arp('ARP/uni2423reply', '00-DE-BB-00-00-00', '127.0.0.1', '00-DE-BB-00-00-00', '127.0.0.1')
    self.assertEqual(stream1.commands, [':TSTReam:TABLe:ADD
', ':TSTReam:TABLe:ID/uni24231
', ':TSTReam:TABLe:ITEM:PROTocol:ARP:OPERation/uni24232
', ':TSTReam:TABLe:ITEM:PROTocol:ARP:SMADdress/uni2423#H00DEBB000000
', ':TSTReam:TABLe:ITEM:PROTocol:ARP:SIADdress/uni2423#H7F000001
', ':TSTReam:TABLe:ITEM:PROTocol:ARP:TIADdress/uni2423#H7F000001
', ':TSTReam:TABLe:ITEM:PROTocol:ARP:TMADdress/uni2423#H00DEBB000000
'])

def test_ipv4_source_address(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    stream1.ipv4_source_address('127.0.0.1/24', 'RANDOM')
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:PROTocol:IP:SA:MASK/uni2423#HFFFFFF00
')
    self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:PROTocol:IP:SA:VALue/#H7F000001
')
    self.assertEqual(stream1.commands[-3], ':TSTReam:TABLe:ITEM:PROTocol:IP:SA:TYPE/uni2423RANDOM
')

def test_ipv4_destination_address(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    stream1.ipv4_destination_address('127.0.0.1/24', 'RANDOM')
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:PROTocol:IP:DA:MASK/uni2423#HFFFFFF00
')
    self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:PROTocol:IP:DA:VALue/#H7F000001
')
    self.assertEqual(stream1.commands[-3], ':TSTReam:TABLe:ITEM:PROTocol:IP:DA:TYPE/uni2423RANDOM
')

def test_ipv6_source_address(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    stream1.ipv6_source_address('1:2:3:4:5:6:7:8/64', 'RANDOM')
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:PROTocol:IPv6:SA:MASK/uni2423#HFFFFFFFFFFFFFFFF0000000000000000
')
    self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:PROTocol:IPv6:SA:VALue/#H10002000300040005000600070008
')
    self.assertEqual(stream1.commands[-3], ':TSTReam:TABLe:ITEM:PROTocol:IPv6:SA:TYPE/uni2423RANDOM
')

def test_ipv6_destination_address(self):
    stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
    stream1.ipv6_destination_address('1:2:3:4:5:6:7:8/64', 'RANDOM')
    self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:PROTocol:IPv6:DA:MASK/uni2423#HFFFFFFFFFFFFFFFF0000000000000000
')
    self.assertEqual(stream1.commands[-2], ':TSTReam:TABLe:ITEM:PROTocol:IPv6:DA:VALue/#H10002000300040005000600070008
')
    self.assertEqual(stream1.commands[-3], ':TSTReam:TABLe:ITEM:PROTocol:IPv6:DA:TYPE/uni2423RANDOM
')

def test_protocol(self):
    for argument_1 in self.argument_1_set:
        stream1 = stream.Stream("1", '1', '5', '1', 'md1230b')
        stream1.protocol(argument_1)
        self.assertEqual(stream1.commands[-1], ':TSTReam:TABLe:ITEM:PROTocol:TYPE/uni2423' + argument_1 + '
')
    del stream1

if __name__ == '__main__':
    unittest2.main()```
F | Python-anritsu documentation

F.1 analyzer API

class Analyzer(address, anritsu_type)
    Control the analyzer via a TCP socket on port 5001 which subsequently controls the generator.

capture(unit, module, port_number)
    Start capturing on the port.

    Parameters
    • unit – the unit (generator) number as a string to be selected
    • module – the module (network card) number as a string to be selected
    • port_number – the port number as a string to be selected

count(unit, module, port_number)
    Start counting on the port.

    Parameter selection – a list of messages to select a port

count_transmit(unit, module, port_number)
    Deducing two functions(count() and transmit()) as one function.

    Parameters
    • unit – the unit (generator) number as a string to be selected
    • module – the module (network card) number as a string to be selected
    • port_number – the port number as a string to be selected

disconnect()
    Disconnects the socket, consequently ending the test.

get_port_counter_group(unit, module, port_number, counter_group=None)
    Get a group of counter values

    Parameters
    • unit – the unit (generator) number as a string to be selected
    • module – the module (network card) number as a string to be selected
    • port_number – the port number as a string to be selected
    • counter_group – the name of the group of counters

port_clear_own(unit, module, port_number)
    Clear the counters and take ownership on the given port.
Parameters

- **unit** – the unit (generator) number as a string to be selected
- **module** – the module (network card) number as a string to be selected
- **port_number** – the port number as a string to be selected

send_msg(messages)

To deduce repetitive code of sending a message.

**Parameter** messages – the list of command messages

send_recv_msg(messages)

To deduce repetitive code of sending a message and waiting for a reply (socket is of blocking type).

**Parameter** messages – the list of query messages

stop_all(unit, module, port_number, when=None, time=None)

Stop all running actions (i.e. counting, transmitting and capturing) on a port.

**Parameters**

- **unit** – the unit (generator) number as a string to be selected
- **module** – the module (network card) number as a string to be selected
- **port_number** – the port number as a string to be selected
- **when** – Choose between CONT or STOP. ‘CONT’ means a continuous stream, a time must be given to the time parameter. ‘STOP’ means wait for the stream to end, then stop all remaining actions.
- **time** – how many seconds to wait for the continuous stream before ending it

stream(stream_identification_number, unit, module, port_number)

Create via the stream module a stream and pass the anritsu type to the stream, because the API is different between different Anritsu device types

**Parameters**

- **stream_identification_number** – set the stream id, must be unique per port
- **unit** – the unit (generator) number as a string to be selected
- **module** – the module (network card) number as a string to be selected
- **port_number** – the port number as a string to be selected

stream_commit(stream_object)

Commit the set stream variables on the Anritsu.

**Parameter** stream_object – a stream object

test_port_counter(unit, module, port_number, counter_name, val1, val2)

Get the port counter values and test if its between the expected range

**Parameters**

- **unit** – the unit (generator) number as a string to be selected
- **module** – the module (network card) number as a string to be selected
- **port_number** – the port number as a string to be selected
- `counter_name` – the counter that needs to be checked
- `val1` – the value where the range starts
- `val2` – the value where the range ends

```python
transmit(unit, module, port_number)
```

Start transmitting on a port.

**Parameters**
- `unit` – the unit (generator) number as a string to be selected
- `module` – the module (network card) number as a string to be selected
- `port_number` – the port number as a string to be selected

```python
wait_for_transmission(unit, module, port_number, when=None, time=None)
```

Stop all running actions (i.e. counting, transmitting and capturing) on a port.

**Parameters**
- `unit` – the unit (generator) number as a string to be selected
- `module` – the module (network card) number as a string to be selected
- `port_number` – the port number as a string to be selected
- `when` – Choose between CONT or STOP. ‘CONT’ means a continuous stream, a time must be given to the time parameter. ‘STOP’ means wait for the stream to end, then stop all remaining actions.
- `time` – how many seconds to wait for the continuous stream before ending it

## F.2 port API

```python
capture(unit_number, module_number, port_number)
```

Creates messages to start counting on a port.

**Parameters**
- `unit` – the unit (generator) number as a string to be selected
- `module` – the module (network card) number as a string to be selected
- `port` – the port number as a string to be selected with ownership taken and its counters and streams cleared

```python
count(unit_number, module_number, port_number)
```

Creates messages to start counting on a port.

**Parameters**
- `unit` – the unit (generator) number as a string to be selected
- `module` – the module (network card) number as a string to be selected
- `port` – the port number as a string to be selected with ownership taken and its counters and streams cleared

```python
initialize(unit, module, port)
```

Creates messages to initialize a port for first use, more specifically: set port to default settings, clear/take ownership, clear/stop counters, clear streams
Parameters

- `unit` - the unit (generator) number as a string to be selected
- `module` - the module (network card) number as a string to be selected
- `port` - the port number as a string to be selected with ownership taken and its counters and streams cleared

`read(counter_name)`

Creates messages to read a port its amount of transmitted frames

Parameter `socket` - the socket which controls the analyzer

`select(unit, module, port)`

Creates messages to select a port

Parameters

- `unit` - the unit (generator) number as a string to be selected
- `module` - the module (network card) number as a string to be selected
- `port` - the port number as a string to be selected

`transmit(unit_number, module_number, port_number)`

Start transmitting on the port.

Parameters

- `unit` - the unit (generator) number as a string to be selected
- `module` - the module (network card) number as a string to be selected
- `port` - the port number as a string to be selected with ownership taken and its counters and streams cleared

F.3 stream API

class Stream(stream_identification_number, unit, module, port, anritsu_type)

This class represents one stream. These functions create command messages and its related query messages. The commands set a variable on the Anritsu. The queries test the user input with the Anritsu current set variable.

Variable commands A list is created for the command messages, which set a variable on the Anritsu

Variable queries A dictionary is created for the query messages, required to test a variable on the Anritsu.

Variable anritsu_type the Anritsu type, because not all Anritsu API messages are applicable on all Anritsu devices

`arp(arp_type, sender_mac, sender_ip, target_mac, target_ip)`

Creates messages to set the ARP protocol

Parameters

- `arp_type` - set the type of ARP as defined in RFC 826 and RFC 903 can select out of the following set: 'ARP request', 'ARP reply', 'RARP request', 'RARP reply'
- `sender_mac` - set the sender's MAC address. (e.g. 00-DE-BB-00-00-00)
• `sender_ip` – set the sender’s IP address. (e.g. 192.168.1.3)
• `target_mac` – set the target’s MAC address. (e.g. 00-DE-BB-00-00-00)
• `target_ip` – set the target’s IP address. (e.g. 192.168.1.3)

`burst_per_stream(burst_per_stream_amount)`
Creates messages to define how many burst are needed for this stream.

**Parameter** `burst_per_stream_amount` – amount of burst

`distribution(stream_distribution_type, jump_to_id=None, count=None)`
Creates messages to define the type of distribution of the stream.

**Parameters**
• `jump_to_id` – if the distribution type is ‘JUMP’, ‘JUMP_COUNT’, ‘JUMP_STOP’, this parameter represents to what stream ID to JUMP to. Expects to be a number between 1 and 256 (requirement: the stream id must be already set on the Anritsu)
• `count` – if the distribution type is ‘JUMP_COUNT’, ‘JUMP_STOP’. This parameter sets the loop count, which is the amount of jumps, before this stream stops. Expects to be a number between 1 and 256.

`ethernet_error(error_type)`
Insert an ethernet error in a stream.


`frame_destination_address(frame_destination_address_hexed_separated, frame_destination_mask_hexed_separated='FF-FF-FF-FF-FF-FF', frame_destination_address_type='STATIC')`
Creates messages to define the frame destination MAC address.

**Parameters**
• `frame_destination_address_hexed_separated` – destination MAC address e.g. ‘00-00-00-00-00-00’
• `frame_destination_mask_hexed_separated` – destination mask MAC address e.g. ‘00-00-00-FF-FF-FF’
• `frame_destination_address_type` – destination address type, can select out of the following set: ‘GATEWAY’, ‘STATIC’, ‘INCREMENT’, ‘DECREMENT’, ‘RANDOM’

`frame_size(frame_size_type, frame_size_value=None, frame_size_maximum=None)`
Creates messages to define what frame size type is needed.

**Parameters**
• `frame_size_type` – the type of frame size, which can be ‘AUTO’, ‘FIXED’, ‘INCREMENT’, ‘RANDOM’
• `frame_size_value` – if the frame size type is ‘AUTO’, this value is ignored. If ‘FIXED’ it is the frame size. If ‘INCREMENT’ or ‘RANDOM’, it is the minimum frame size.
• `frame_size_maximum` – ignored with frame size type ‘AUTO’ and ‘FIXED’. If it is ‘INCREMENT’ or ‘RANDOM’ it is the maximum frame size.
Python-anritsu documentation

Appendix F

frame_source_address(frame_source_address_hexed_separated, frame_source_mask_hexed_separated='FF-FF-FF-FF-FF-FF', frame_source_address_type='STATIC')

Creates messages to define the frame source MAC address.

Parameters
- frame_source_address_hexed_separated – source MAC address e.g. '00-00-00-00-00-00'
- frame_source_mask_hexed_separated – source mask MAC address e.g. '00-00-00-FF-FF-FF'
- frame_source_address_type – source address type, can select out of the following set: ‘GATEWAY’, ‘STATIC’, ‘INCREMENT’, ‘DECREMENT’, ‘RANDOM’

frames_per_burst(frames_per_burst_amount)

Creates messages to define how many frames are needed for every burst.

Parameter frames_per_burst_amount – amount of frames

inter_burst_gap(inter_burst_gap_value)

Creates messages to define the Inter Burst Gap (IBG).

Parameter inter_burst_gap_value – the IBG in ns.

inter_frame_gap(inter_frame_gap_type, inter_frame_gap_value, inter_frame_gap_maximum=None)

Creates messages to define the Inter Frame Gap (IFG).

Parameters
- inter_frame_gap_type – the type of IFG, which can be ‘FIXED’ or ‘RANDOM’.
- inter_frame_gap_value – if the IFG type is ‘FIXED’, this will be the fixed IFG in nanosecond (ns). If its ‘RANDOM’, the keyword represents the minimum time in ns.
- inter_frame_gap_maximum – if the IFG type is ‘RANDOM’, this keyword is the maximum time in ns.

inter_stream_gap(inter_stream_gap_value)

Creates messages to define the Inter Stream Gap (ISG).

Parameter inter_stream_gap_value – the ISG in ns.

ipv4_destination_address(ipv4_destination_address, ipv4_destination_address_type='STATIC')

Creates messages to define the packets destination IPv4 address.

Parameters
- ipv4_destination_address – destination IPv4 address e.g. '127.0.0.1/24'
- ipv4_destination_address_type – destination address type, can select out of the following set: ‘GATEWAY’, ‘STATIC’, ‘INCREMENT’, ‘DECREMENT’, ‘RANDOM’

ipv4_error(error_type)

Insert an IPv4 error in a stream.

Parameter error_type – what type of IPv4 error, can select out of the following set: ‘CHECKSUM’.

ipv4_source_address(ipv4_source_address, ipv4_source_address_type='STATIC')

Creates messages to define the packets source IPv4 address.

Parameters
• **ipv4_source_address** – source IPv4 address e.g. ‘127.0.0.1/24’
• **ipv4_source_address_type** – source address type, can select out of the following set: ‘GATEWAY’, ‘STATIC’, ‘INCREMENT’, ‘DECREMENT’, ‘RANDOM’

`ipv6_destination_address(ipv6_destination_address, ipv6_destination_address_type='STATIC')`

Creates messages to define the packets destination IPv6 address.

**Parameters**
• **ipv6_destination_address** – destination IPv6 address e.g. ‘1:2:3:4:5:6:7:8’
• **ipv6_destination_address_type** – destination address type, can select out of the following set: ‘GATEWAY’, ‘STATIC’, ‘INCREMENT’, ‘DECREMENT’, ‘RANDOM’

`ipv6_source_address(ipv6_source_address, ipv6_source_address_type='STATIC')`

Creates messages to define the packets source IPv6 address.

**Parameters**
• **ipv6_source_address** – source IPv6 address e.g. ‘1:2:3:4:5:6:7:8/64’
• **ipv6_source_address_type** – source address type, can select out of the following set: ‘GATEWAY’, ‘STATIC’, ‘INCREMENT’, ‘DECREMENT’, ‘RANDOM’

`protocol(stream_protocol)`

Creates messages to define the protocol in the stream.

**Parameter**  

`test_frame(test_type, length_or_offset=None, flow_id=None)`

Adds a test frame in the frame its data field

**Parameters**
• **test_type** – the type of test used, can select out of the following set: ‘PRBS’, ‘flow’.
• **length_or_offset** – the length in bytes of a PRBS test frame and the offset in bytes of a flow test frame.
• **flow_id** – when a flow test frame is selected, this is the ID.

### F.4 input_validator API

`string_set(allowed_strings, user_input)`

This function is used to validate user input with the allowed strings for the targeted Anritsu variable. If this function returns True, the user input is validated.

**Parameters**
• **allowed_strings** – an array of allowed strings which the Anritsu should allow
• **user_input** – the user input its string to test against the allowed list of strings
value_minimum_maximum(minimum, maximum, user_input)

This function is used to validate user input with the allowed value range for the targeted Anritsu variable. If this function returns True, the user input is validated.

Parameters

- minimum – the minimum allowed integer
- maximum – the maximum allowed integer
- user_input – the user input its integer for testing if its between the minimum and maximum range

F.5 iptoheX API

IPtoHex(ip_input)

Takes an IP address as a string and returns it in a uppercase hexadecimal with a syntax prefix (as required by the Anritsu).

Parameter ip_input – a string that represents an IP address (e.g. IPv6: ‘fe80::dead:beef/64’ or IPv4: ‘127.0.0.1/24’)

F.6 mactoheX API

MactoHex(dec_mac, unit=None, module=None, port=None)

Takes a MAC address as a string and returns it in uppercase a hexadecimal form with a prefix (as required by the Anritsu). When only the last 2 octects of a MAC address is given, the unit, module and port number is used as the prefix. This should only be used for the source address.

Parameters

- dec_mac – a string that represents a MAC address (e.g. ’00-00-FF-FF-FF’ or ’FF-FF’)
- unit – the unit (generator) number as a string to be selected
- module – the module (network card) number as a string to be selected
- port – the port number as a string to be selected

F.7 error API

exception AnritsuCommandError

Exception raised when an expected variable value is not set on a Anritsu after a commit.

Variable command The message which contains the variable
Variable sent_value The value we have sent and expected to be set on the Anritsu
Variable anritsu_value The variable value the Anritsu returns it has set.

exception AnritsuQueryError

Exception raised when an expected variable value is not set on a Anritsu after a commit.
**Variable query**  The message which contains the variable

**Variable sent_value**  The value we have sent and expected to be set on the Anritsu

**Variable anritsu_value**  The variable value the Anritsu returns it has set.

**exception AnritsuSupported**  
Exception raised when a requested feature is not supported.

**Variable command**  The message which contains the variable

**Variable sent_value**  The value we have sent and expected to be set on the Anritsu

**Variable anritsu_value**  The variable value the Anritsu returns it has set.

**exception AnritsuTimeout**  
Exception raised when an expected variable value is not set on a Anritsu after a commit.

**Variable command**  The message sent before a timeout occurred

**exception Error**  
Base class for exceptions in this module.
AMS-IX Amsterdam Internet Exchange
HvA Hogeschool van Amsterdam
IXP Internet Exchange Point
NOC Network Operations Center
GUI Graphical User Interface
EFT Early Field Trail
GPIB General Purpose Interface Bus
UML Unified Modeling Language
OOP Object-Oriented Programming
CAT Customer Acceptance Testing
PXC Photonic Cross-Connect
TCP/IP Transmission Control Protocol/Internet Protocol
TCP Transmission Control Protocol
OSI Open Systems Interconnection
BGP Border Gateway Protocol
VLAN Virtual Local Area Network
SNMP Simple Network Management Protocol
SSH Secure Shell
HTML HyperText Markup Language
Euro-IX European Internet Exchange Association
CERN European Organization for Nuclear Research
ISPs Internet Service Providers
IP Internet Protocol
IPv4 Internet Protocol version 4
IPv6 Internet Protocol version 6
SARA Stichting Academisch Rekencentrum Amsterdam
MPLS Multi Protocol Label Switching
VPLS Virtual Private LAN Service
LSP Label Switched Paths
LDP Label Distribution Protocol
PE Provider Edge
CE Customer Edge
GE Gigabit Ethernet
10GE 10 Gigabit Ethernet
100GE 100 Gigabit Ethernet
FPGA Field-Programmable Gate Array
GPP General Purpose Processor
AS Autonomous System
ASN Autonomous System Number
IANA Internet Assigned Numbers Authority
IEEE Institute of Electrical and Electronics Engineers
PEP Python Enhancement Proposal
IM Instant Messaging
H  |  References


