Haaga-Helia
ammattikorkeakoulu Oy

# Automation of Router Configuration

Joram Puumala

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2017

Abstract

**Author(s)**
Joram Puumala

**Program**
Tietojenkäsittelyn koulutusohjelma

| **Subject of Thesis** | **Page count (+attachment pages)** |
| --- | --- |
| Automation of Router Configuration | 51 + 7 |

The goal of this thesis was to find out a fast, automated and efficient way to configure Advantech B+B's Mobile Routers and how much time can be saved vs. manual configuration.

In the theory part, different configurations methods are mentioned. The methods that can be applied to this particular case, are explained in more detail. The theory part also provides general information about configuration management and configuration management systems.

In the code-building part, it is shown how the actual code was built. It is not only about building the code though, it also includes explanations why something is being done and why something should be avoided. Code snippets are associated with appropriate screenshots. The screenshots help in understanding what happened when a code snippet was run.

Results provided clear answers to research questions. Based on the results, it can be concluded that there is a better way to configure Advantech B+B's mobile routers than manual configuration. The automation method proved to be lots of faster and didn't make misconfigurations during the process.

The thesis ends with a discussion about future development, conclusion and comments from the CEO of NDC Networks Oy.

**Key words**
Python, Configuration, Router, Automation, Excel, SSH

# Table of Contents

# 1 Introduction

Once an independent IT equipment importer, NDC has had to look for new ways to hang on to customers. One succesful strategy has been to preconfigure customer equipment such as routers before the equipment are shipped from NDC to the end customer. The strategy has been so succesful that now NDC is in trouble. Router orders are getting bigger, new customers and services are stirring the soup. NDC Networks is facing a serious problem, they are running out of resources. Router configuration manually is no longer an option. Hiring new employees to configure routers would be very expensive. There must be a better and a cheaper option.

## 1.1 Background

NDC Networks, founded in 1993, is a small Finnish company based in Espoo. There are currently eight people working at NDC, but the company is expected to grow since its revenue have gone up steeply this year (2017). NDC's revenue in 2016 totaled €2 million, and in November 2017 the company's revenue has already passed the €3 million mark. Also, NDC's operating income saw a major growth in 2016, as it went up from €90 000 (2015) to €216 000. Over a two year span, the growth was €205 000 (Operating income in 2014 was 9 000 euros) (Finder, 2017). The positive trend is expected to continue.

The company is known for its expertise in networks and Virtual Private Network (VPN) management. Routers that specialists at NDC Networks configure are mostly mobile routers. These mobile routers are used in different environments such as power utilities and industrial companies. Different mobile routers do exist, but typically they are connected to a mobile network using a traditional Subscriber Identity Module (SIM) card and are capable of changing their point of attachment to the Internet, moving from one link to another link (Ernst & Lach, 2007, 5). Because of the possible different features and network specific requirements, configurations are usually unique for each customer.

## 1.2 About this thesis

This thesis is about finding a fast, automated and efficient way to configure Advantech B+B's mobile routers according to customer specifications. The expected outcome is a Python(v3) program that is able configure the router and update a customer specific excel file. The routers use Linux (Kernel 3.12.10+) operating system, and the Python version 3.5.2 will be used to run the program. The routers themselves don’t support any version of Python language, so the program will use Python’s ability to convey shell commands to the routers and execute them. Two non-native Python modules are needed as well, paramiko (2.3.1) and openpyxl (2.3.0).

There are lots of code examples throughout the thesis. The idea is to build the program step by step and explain what is happening and why something is being done. Because the program will be built step by step, each function will be tested separately, associated with appropriate screenshots. In the end when everything seems to work as expected, all the code will be put together and tested. Source code of the program can be found in Appendix.

The thesis can be divided into two parts. The first part will give background information about router configuration, what different configuration methods exist and how those differ from one another. The second part is about building the code and testing it, and it will be concluded with results and ideas for future development, a final conclusion and comments on the tool from the CEO of NDC Networks.

### 1.3 Goals of the study

Main goal of this study is to find out how to speed up the router configuration process.

Research questions are:

- Is there a better way to configure a router than manually, particularly in this case?
- How much time can be saved, automation vs manual?
- How many fewer errors/misconfigurations will occur?

## 2 Router configuration, management and techniques

Routers can be configured in different ways, depending on model and manufacturer. Some routers have a web interface, while other routers can be configured only by using text-based command line. Router configuration using a web interface is usually pretty straightforward, since Web interfaces are designed so that even laymen have some clue how to configure a router. Router configuration on a command line can be a little trickier. Not only because some knowledge of how a command line works is needed, but because commands may change radically between different router operating systems. For example, Cisco, which is dominates the router market with a market share of 55.1% (IDC, 2017), has its own Cisco IOS operating system with its own unique commands. Similarly, Huawei's routers run their own operating system, which means different commands apply when configuring Huawei routers.

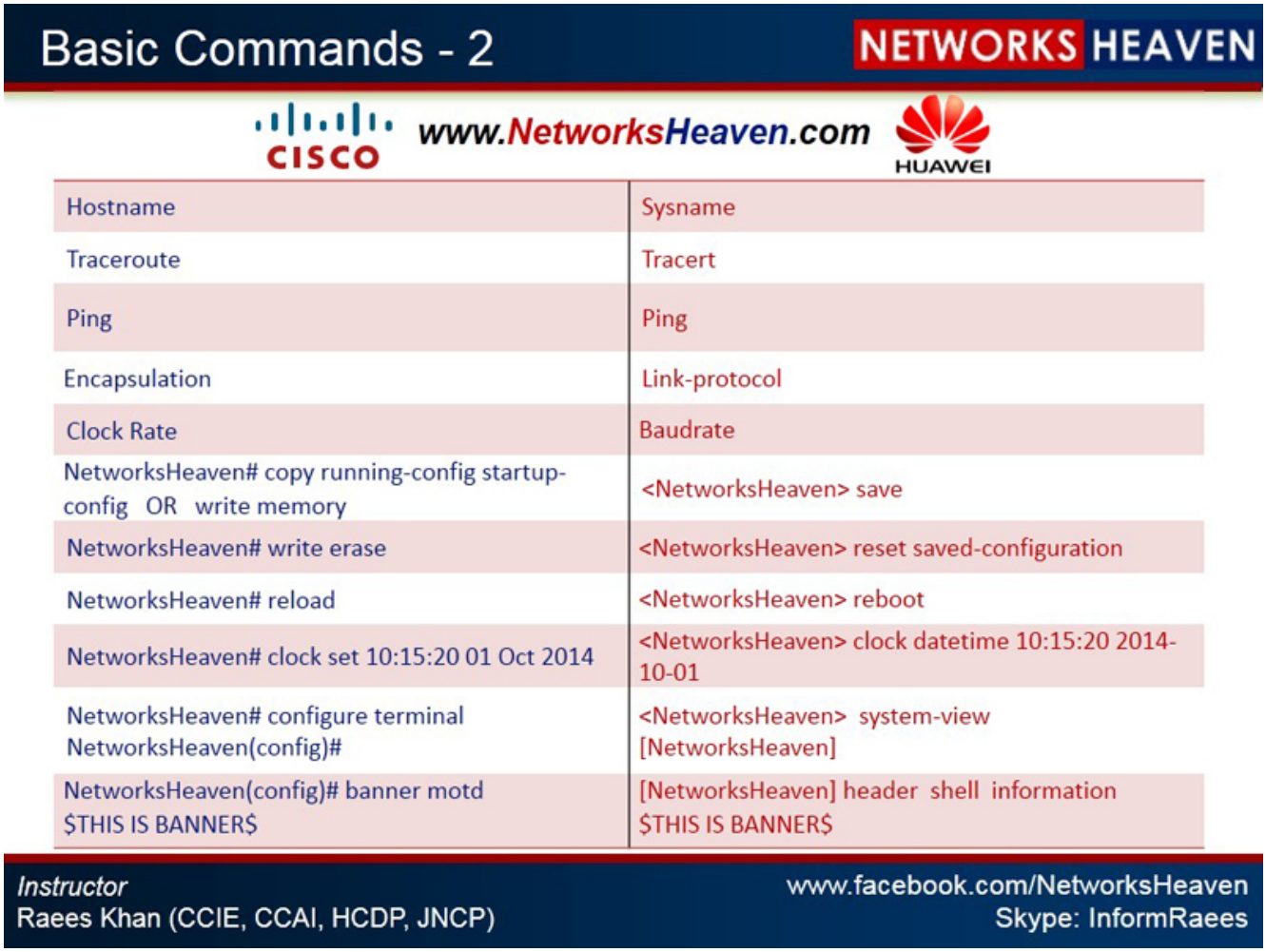


*Fig. 1 - Cisco vs Huawei, Basic Commands - 2 (Khan, 2014)*

Since routers can be configured using a web interface or a command line, the configuration process can be automated. This is because lots of tools exist that make it possible. Many vendors also allow scripting on the command line, so this is a huge first step towards automation. Sometimes that is not enough. Not everything can be done or added via a configuration/script file, for example some external modules may need to be added by hand. This means manual labor and additional time. The good thing is, it can be automated using external methods. There are different ways to automate such tasks. One way to do it is over an SSH connection, using Python for example. This technique falls under the command line configuration technique. Also, one possibility is to create an automation tool/robot that uses router's Web user interface. This technique is called Web scraping (Heydon & Najork, 1999).

There are still some other ways to configure a router as well. One could use a centralized server which provides configurations to routers. This of course means that there should be an initial configuration file inside the router so it knows where to connect to in order to get the configuration. Another way is to use a USB device. When the USB device is plugged into a router, the configuration file can be downloaded and run by the router. This thesis is not going to cover these two methods in more depth, because neither of the methods can prove to be helpful in this case. There are multiple reasons why. Firstly, not all the router models have USB ports, and the ports might not be configured to automatically retrieve and run the configuration file when the USB device is plugged. It is also possible that server configuration is disabled by default. Both USB configuration and server configuration are disabled by default in Advantech's routers, as can be seen in Fig. 2.

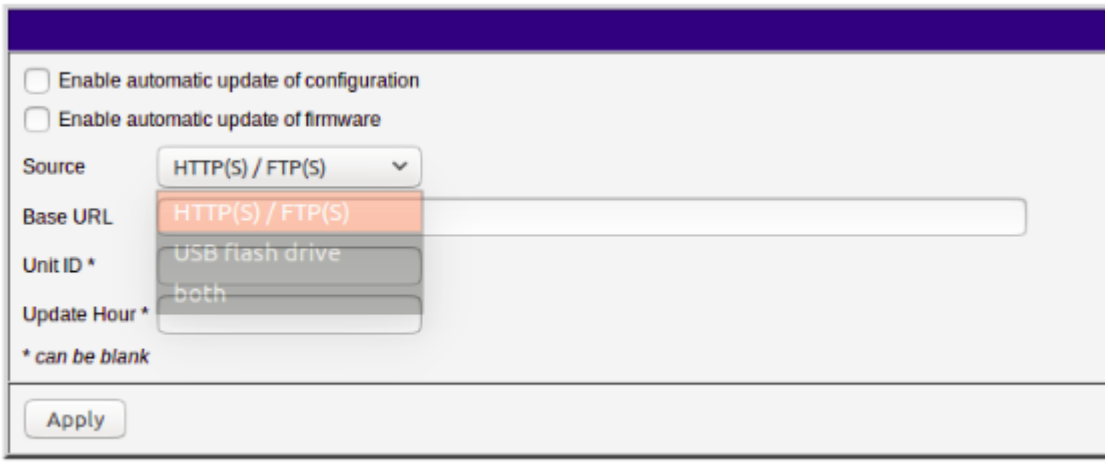


*Fig. 2 - Automatic updates from USB devices and HTTP addresses are disabled by default*

## 2.1 Configuration management

Configuration Management (**CM**) is a discipline applying administrative and technical direction and surveillance to identify and document functional and physical characteristics of a configuration item. It is also about identifying control changes of those characteristics, recording and reporting change processing and implementation status, and verifying compliance with requirements specified (IEEE , 1990). Definition of CM also includes terminology such as baseline, release and version (Dart, 1991). One of the core functions of CM is automation. Automation has many benefits over manual configuration, since manual configuration practices are limited in many ways. For example, manual configuration is costly, time-consuming and unscalable. Now imagine having tens of thousands of network elements, and applying a new configuration to every single one by hand. It would be pretty much impossible and eat way too much resources. Also, manual configuration is prone to misinterpretations and errors. Engineering guidelines can be ambiguous, sometimes even imprecise and this leads to multiple interpretations (Enck et al., 2007).

### 2.2 Configuration management systems

There is no one unified definition of a CM system. Consider this, if there's version control in a system, is it a CM system? Any system providing some form of system structuring, system modelling, configuration identification, version control and is intending to be a CM system to some degree, is considered to be a CM system by the software engineering community. Also, it is important to identify the difference between a CM system and a CM tool. A CM tool is rather a stand-alone tool than a system, because the CM tool is to be installed into an existing environment (Dart, 1991). One of the core functions of CM systems is to coordinate access to a common set of artifacts by multiple developers/administrators, working on the same project. Ideally project management assings tasks which are mutually exclusive, but the reality is changes made by one administrator or developer affect other's work (Sarma, et al., 2003). There are plenty of CM systems and tools available these days.

### 2.3 Web scraping

Extracting patterned data from web pages in the Internet is called Web scraping. There are different uses for Web scraping, one major use is for businesses to track pricing activities of their competitors. Web scraping is an efficient method for reducing manual labor time and labor can be saved in massive amounts. Web scraping can prove itself useful in the realm of research as well (Haddaway, 2015). Web scraping is one way to create an automation tool. It requires a Web user interface for router configuration.

The basic idea behind a Web scraping framework is that it establishes communication with a user defined Web page using the HTTP protocol, which is a stateless text-based Internet protocol designed to coordinate the request-response transactions between a client and a Web server, in which the cli-

ent is typically a Web browser. The "User-Agent" header also plays a big role, because it tells the server whether it is trying to be accessed by a robot or a browser. Once the Web scraping framework has retrieved the HTML documents using GET method, contents of interest can be extracted. Because extracting the contents of interest is relevant, regular expressions alone or with a combination of additional logic prove to be powerful and thus are widely adapted. Alternative methods include selector-based languages such as XPath and the CSS selector syntax (Glez-Peña et al., 2014).

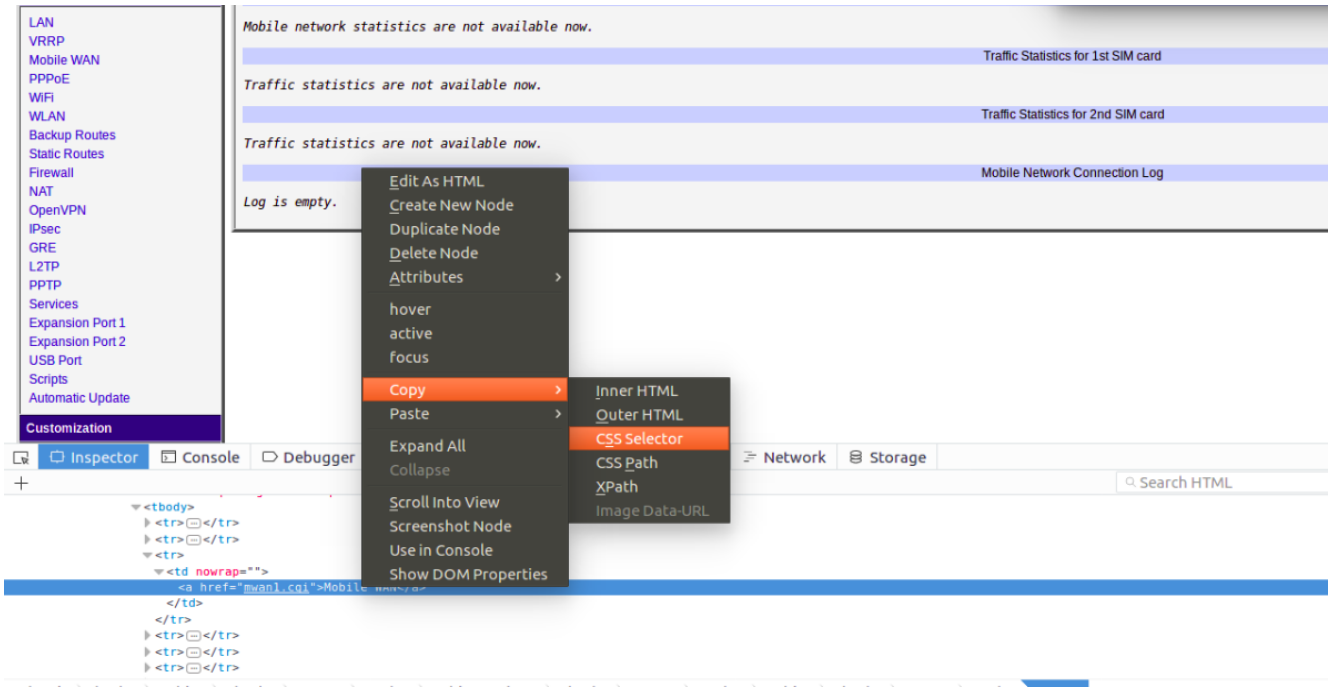


*Fig. 3 - Copying CSS selector*

Web scraping has some downsides too. The performance of different Web scraping frameworks differ between different approaches. It can be time consuming to find the best solution in a particular case. For example, Beautiful Soup (pure Python) is way slower than scraping approach with Python's regular expressions module (written in C) or Lxml, which is also written in the C programming language (Lawson, 2015). Also, Web pages do change. This may break one's code and it has to be fixed before it works again. What is more, the information available may be insufficient (Griffioen, et al. 2014).

### 2.4 Command line configuration

Configuring a router via command line is the "traditional way", as it is the primary user interface still in use today (Cisco, 2013). Every router has the command line option, but not every router has a Web user interface, which is one reason why the command line is still preferred. Router configuration on a command line is basically just running a string of commands to change the way a router behaves. Configuration commands usually differ depending on manufacturer.

As mentioned earlier, many router models have the option to write configuration scripts. Cisco is a good example.
Cisco IOS scripting with Tcl is a popular thing, since it would make no sense to run commands one by one. To be able to write scripts for Cisco IOS, one needs to be familiar with Cisco IOS command line commands and Tcl programming (Cisco, 2014). With a Linux based router, Bourne Shell (sh) or Bourne again Shell (bash) may be present, which allows Shell Scripting.

Sometimes something external may be needed. For example, some configuration files and modules need to be transferred to router. Maybe some settings that couldn't be included inside the configuration file/script. A dream situation is that a configuration management system manages it later, but unfortunately that is not always the case. Luckily, many routers ship with SSH (configurable or configured), and external scripting can be used over an SSH connection.

# 3 Current state

There are currently lots of problems in NDC's router configuration. The greatest of the problems is that everything is done manually using router's graphical Web user interface. Additionally, after having finished router configuration in the web GUI, a specific customer excel file needs to be updated with information such as router's serial number, MAC address, IP address and model. This is manual labor as well.

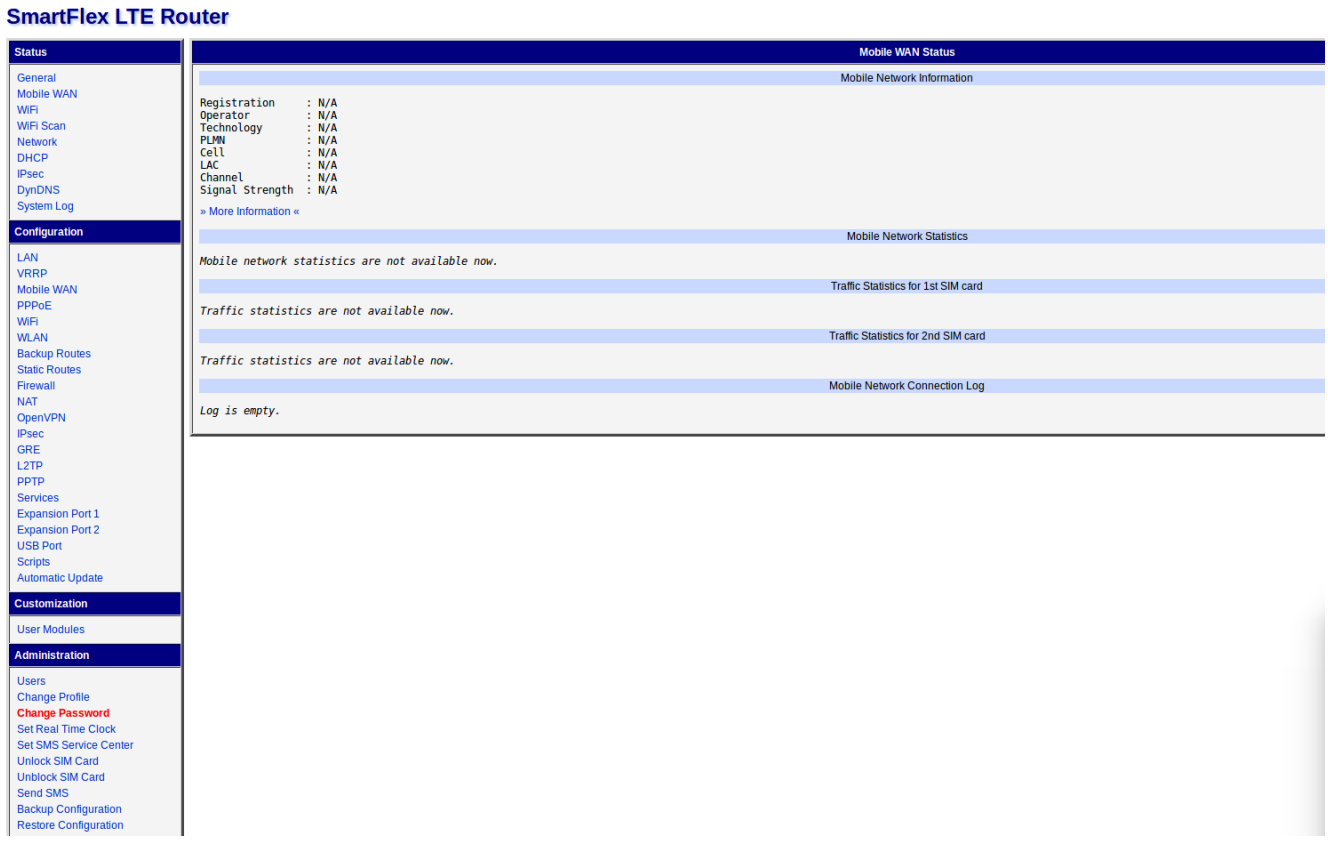


*Fig. 4 – SmartFlex Mobile Routers' Web Interface*

## 3.1 The current process of router configuration

Below is a real process of manual router configuration for one customer. Each customer's process is little different. They may want different user modules and some information will differ such as passwords. Anyway, the idea is the same.

1. Plug in the router
2. Browse to its default IP address
3. Log in using username and password

4. Click Restore Configuration
5. Click Add Configuration and browse to the right file
6. Click Add or Update
7. After confirmation takes you to another page, click Back
8. Click Services
9. Click SNMP under Services
10. Change SNMP name
11. Click Apply
12. Click Back
13. Click Change password
14. Type a new password twice
15. Click Apply
16. Click Back
17. Click User Modules
- Currently two user modules are added
18. Click Add new and browse to the right file
19. Click Add or Update
20. Click Back
21. Repeat 18.
22. Repeat 19.
23. Repeat 20.

The process includes lots of clicking and browsing to files, which obviously takes time. Also, chances are that a person congifuring the router does something wrong and the process has to be started over. Even worse, an imperceptible mistake during the process happens which has to be debugged and fixed later when found.

### 3.2 The current process of updating Excel

When the actual configuration part is ready, a specific customer excel file has to be updated. Below are the current values which will be written into the file of the example customer.

1. Router's VPN IP address
2. VPN IP address' netmask
3. Router's serialnumber
4. Router's MAC address
5. Router's model
6. Date

7. Sales reference

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | VPN IP | Network Mask | SerialNo | MAC | Model | Date | SWH ok | Zabbix ok | Zabbix alarms | Reference |
| 2 | 10.240.7.1 | 255.255.255.0 | 4932538 | 00:0A:14:54:65:AC | RT3G-310-W | 03/10/17 | | | | MT102017 |
| 3 | 10.240.8.1 | 255.255.255.0 | 4905974 | 00:0A:14:54:65:BB | RT3G-310-W | 03/10/17 | | | | MT102017 |
| 4 | 10.240.9.1 | 255.255.255.0 | 5955936 | 00:0A:14:54:65:88 | RT3G-310-W | 03/10/17 | | | | MT102017 |
| 5 | 10.240.10.1 | 255.255.255.0 | 4952635 | 00:0A:14:54:65:6B | RT3G-310-W | 03/10/17 | | | | MT102017 |
| 6 | 10.240.11.1 | 255.255.255.0 | 5962967 | 00:0A:14:54:65:AA | RT3G-310-W | 03/10/17 | | | | MT102017 |
| 7 | 10.240.12.1 | 255.255.255.0 | 2438521 | 00:0A:14:54:65:AD | UR5i V2 | 03/10/17 | | | | MT102017 |
| 8 | 10.240.13.1 | 255.255.255.0 | 2534559 | 00:0A:14:54:65:1B | UR5i V2 | 03/10/17 | | | | MT102017 |
| 9 | 10.240.14.1 | 255.255.255.0 | 6538458 | 00:0A:14:54:65:D5 | UR5i V2 | 03/10/17 | | | | MT102017 |
| 10 | 10.240.15.1 | 255.255.255.0 | 3328530 | 00:0A:14:54:65:BC | UR5i V2 | 03/10/17 | | | | MT102017 |
| 11 | 10.240.16.1 | 255.255.255.0 | 4328357 | 00:0A:14:54:65:B7 | UR5i V2 | 03/10/17 | | | | MT102017 |
| 12 | 10.240.17.1 | 255.255.255.0 | 3348574 | 00:0A:14:54:65:E6 | UR5i V2 | 03/10/17 | | | | MT102017 |
| 13 | 10.240.18.1 | 255.255.255.0 | 5348261 | 00:0A:14:54:65:DD | UR5i V2 | 03/10/17 | | | | MT102017 |
| 14 | | | | | | | | | | |
| 15 | | | | | | | | | | |
| 16 | | | | | | | | | | |
| 17 | | | | | | | | | | |
| 18 | | | | | | | | | | |
| 19 | | | | | | | | | | |
| 20 | | | | | | | | | | |
| 21 | | | | | | | | | | |
| 22 | | | | | | | | | | |
| 23 | | | | | | | | | | |

*Fig. 5 - Excel file in which the information above is stored*

# 4 Automating the process

Configuring thousands of routers manually is time consuming and tedious. Humans also tend to make mistakes. A good way to get rid of possible misconfigurations and speed up the process is automating it.

The automation program is developed specifically for Advantech B+B's mobile routers. These routers, run Linux (Kernel 3.12.10+) operating system with embedded BusyBox software. The Linux operating system doesn't have a package manager. Even though this automation program is designed for Advantech's mobile routers, with minor changes it can be used for other Linux based routers as well.

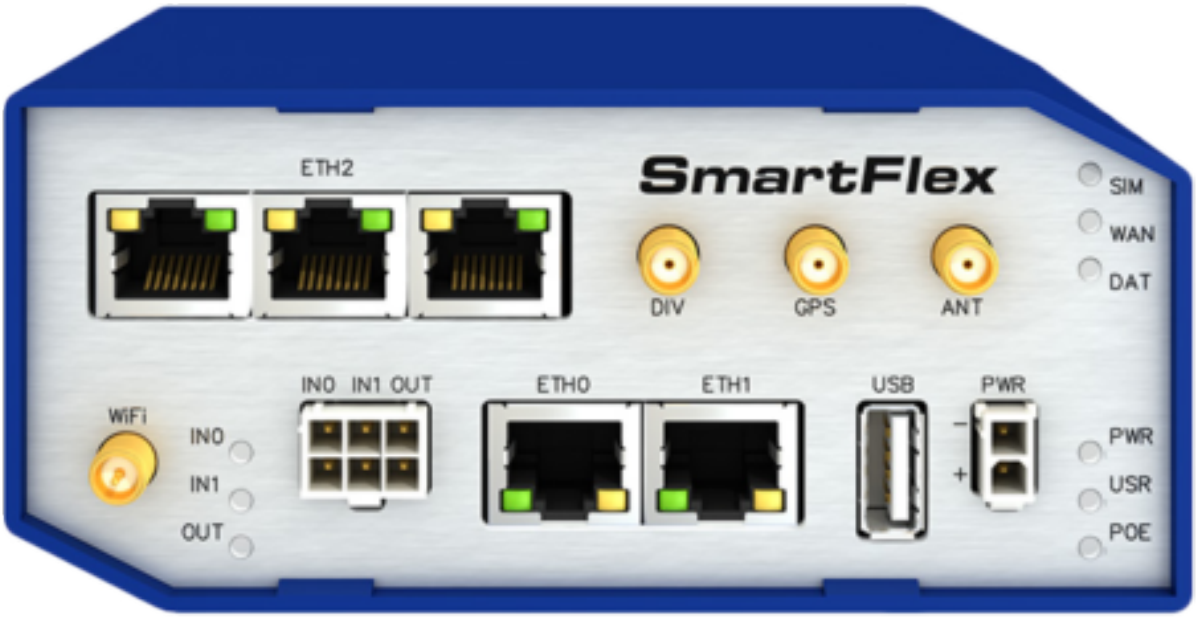


*Fig. 6 - Advantech B+B's SmartFlex mobile router (Advantech B+B, 2017)*


*Fig. 7 - Advantech B+B's SmartStart LTE mobile router (Advantech B+B, 2017)*

The program will use a command line configuration technique over an SSH (Secure Shell) connection, which it initiates when the program is started. Language of choice is Python (3.5.2), because of its versatility, efficiency and simplicity. Ideally, the program can be run on different operating systems, such as different Linux flavors and Windows versions. The program could also be made using a Web scraping framework, such as Selenium. This option can be deemed dirty and code may be fragile, as it can break when the Web interface has changed only very little. This can occur e.g. when a firmware update to the router introduces new features that are accessible via new options in the web interface. So, it is a better idea to stick with the command line. This configuration is just a one-time process, since later the router can be managed by using Advantech's management system (SmartWorx Hub) and other external scripts using the management system's API.

### 4.1 Functions and configuration order

Before writing the actual code, it is important to know what functions have to be developed and in which order they should be placed. For example, router's new configuration file has to be in place in the router before changing its SNMP name, because the new configuration file will overwrite SNMP settings including SNMP name.

Order of functions:

- 1. Initialize SSH connection
- 2. Fetch router's serial number and MAC address
- 3. Put new configuration file into router and run it
- 4. Change SNMP name
- 5. Add user modules
- 6. Change password
- 7. Download backup
- 8. Update excel sheet

- Additionally, for each task, functionality will be written to confirm the success of configuration.

### 4.2 Initializing SSH connection to router

First, a connection needs to be established between a configuring computer and a router. The computer and the router are connected with an ethernet cable. So, the first step is to write a code snippet that initializes the connection over SSH. Python module *paramiko* will be used, which is a non-native Python module.

```
import paramiko

#default IP for the routers is always the same
router_dflt_ip = "192.168.1.1"
uname = "root"
passwd = "Password3xample-"

#defining the SSH connection
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(router_dflt_ip, username=uname, password=passwd)
```

The command "ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())" is important here because it automatically deals with host keys. To demonstrate what happens without this line, it will be removed for a test run.

```
Traceback (most recent call last):
  File testing.py, line 8, in <module>
    ssh.connect(router_dflt_ip, username=uname, password=passwd)
  File /usr/local/lib/python3.5/dist-packages/paramiko/client.py, line 395, in connect
    self, server_hostkey_name, server_key
  File /usr/local/lib/python3.5/dist-packages/paramiko/client.py, line 752, in missing_host_key
    raise SSHException('Server %r not found in known_hosts' % hostname)
paramiko.ssh_exception.SSHException: Server '192.168.1.1' not found in known_hosts
```

*Fig. 8 - Running the program without "ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())"*

As can be seen in Figure 8, "paramiko.ssh_exception.SSHException" was raised. This is because there is a missing host key. Now running the original program should produce different result, because it knows how to deal with the host key.

```
$ python testing.py
$
```

*Fig. 9 - No exceptions are raised this time*

This time the program runs without any errors. This means the SSH connection was succesfully established.

### 4.3 Fetching router's serial number and MAC address

In order to fetch a router's serial number, the first thing is to know how to find it inside the router. Fortunately, command "status -v sys" exists. This command prints too much information though, so command line tools *grep* and *awk* can be used to get just what is needed.

```
# status -v sys
Product Name     : SPECTRE-v3L-LTE
Firmware Version : 6.1.1 (2017-03-09)
Serial Number    : 6600154
Profile          : Standard
Supply Voltage   : 11.8 V
Temperature      : 41 ◆C
CPU Usage        : 95%
Memory Usage     : 25996 KB / 511388 KB
Time             : 2017-10-19 09:45:03
Uptime           : 0 days, 17 hours, 56 minutes
#
```

*Fig. 10 - status -v sys command run on router's command line*

Now when concatenating *grep* and *awk* to the command, the first thing is to *grep* for "Serial Number" to get the correct line, after which *awk*'s print function can be used to print the correct column. The final command is as follows, ”*status -v sys |grep "Serial Number" |awk '{print $4}'*".

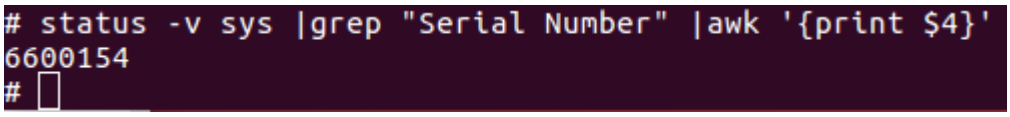


*Fig. 11 - This time, only serial number is printed out to standard output*

Knowing how to get router's serial number on the command line, it is time to integrate it with the python program.

```
import paramiko

#function runs command inside cmd variable on router's command line
#standard output is read and the function returns serial number
def get_serial():
    cmd = "status -v sys |grep \"Serial Number\" |awk '{print $4}'"
    ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
    outp = ssh_stdout.readlines()
    serial = outp[0].strip()
    return serial

router_dflt_ip = "192.168.1.1"
uname = "root"
passwd = "Password3xample-"

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(router_dflt_ip, username=uname, password=passwd)

#these two lines below are just to confirm that the function
#works as expected.
serial = get_serial()
print(serial)

ssh.close()
```

First the command that prints router's serial number is put into variable cmd. When the command is run on router's command line, all of its output will be stored in ssh_stdout variable. Variable outp is used to store values of ssh_stdout in a tuple, after which the first item that is the serial number is chosen.

```
joram@joram:~/opparitesting$ python testing.py
6600154
```

*Fig. 12 - Serial number is returned*

It works as expected. Also, now it can be confirmed that an SSH connection was succesfully established between the computer and the router. Another very similar function needs to be created, but this time MAC address of eth0 port needs be returned. The MAC address of router's eth0 port can be found running "*ifconfig eth0*" command. Once again, the command gives too much information, so *grep* and *awk* will be used again.

```
eth0      Link encap:Ethernet  HWaddr 00:0A:14:84:DF:88
          inet addr:192.168.1.1  Bcast:192.168.9.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1111 errors:0 dropped:0 overruns:0 frame:0
          TX packets:778 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:98317 (96.0 KB)  TX bytes:129482 (126.4 KB)

#
```

*Fig. 13 - Router's eth0 interface*

Concatenation of *grep* and *awk* to the original command will do the job; "*ifconfig eth0 |grep "HWaddr" |awk '{print $5}*'".

```
# ifconfig eth0 |grep "HWaddr" |awk '{print $5}'
00:0A:14:84:DF:88
```

*Fig. 14 - Only MAC address of eth0 interface is printed out on the screen this time*

So now a Python function that fetches router's MAC address can be written. Because the function will not differ that much from the get_serial() function, the structure can be copied as it is enough to change the command in variable cmd..

```
import paramiko

def get_mac():
        cmd = "ifconfig eth0 |grep \"HWaddr\" |awk '{print $5}'"
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
        outp = ssh_stdout.readlines()
        mac = outp[0].strip()
        return mac

router_dflt_ip = "192.168.1.1"
uname = "root"
passwd = "Password3xample-"

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(router_dflt_ip, username=uname, password=passwd)

mac = get_mac()
print(mac)

ssh.close()
```

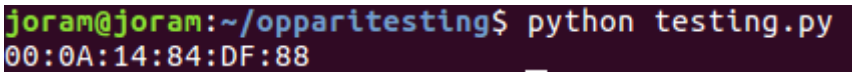


*Fig. 15 - Function get_mac() returns MAC address of router's eth0 interface*

This time, when the Python program was run, MAC address was returned as expected. Now there are two working functions that fetch two values of high importancy. Both of the values will be used later in the program. For example, router's current SNMP name will be changed to router's serial number.

### 4.4 Restoring router's configuration

This is the first phase in which actual changes to router's current configuration are made. Every single router has its own unique configuration file. What makes the file unique are certificates and its VPN IP address. The VPN IP address is also used in the filename when it is created by NDC's server. For example, a typical configuration filename could be "customer_10.240.254.cfg". Now because the file-

name is unique for every router, it would be a bad idea to put it inside the Python program. It would be time consuming and additional work to change it everytime before the program is run. So, instead of putting the filename inside the code, it will be given as a parameter, so that one number in the file name can easily be changed before re-running the program. To be able to give parameters to the python program, "*sys*" module will be imported.

The syntax for the actual restore command inside the router is as simple as "*restore <filename>*". But before anything can be restored, a configuration file has to be transferred to the router. Once the file is in the router, "*restore <filename>*" can be run. It is also a good practice to make sure that the command ran succesfully. If it did, "Configuration succesfully updated." will be printed to standard output. It is possible to use this information to check whether everything went well or awry.

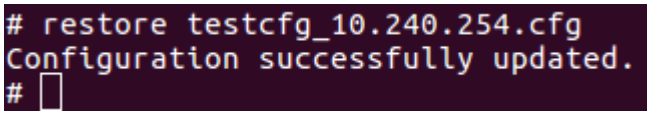


*Fig. 16 - "Configuration succesfully updated." indicates success*

Now everything is pretty straightforward, so it can be put into the Python program.

```
import paramiko
import sys

#takes one parameter, which should be configuration file
#sftp is used to transfer the file to router
#check is made if restore command was succesfully run, return OK or FAILED
def restore_cfg(restore_file):
    orig = restore_f
    dest = "/root/" + restore_file
    cmd = "restore " + dest
    sftp = ssh.open_sftp()
    sftp.put(orig, dest)
    ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
    success = "Configuration successfully updated."
    outp = ssh_stdout.readlines()
    status = outp[0].strip()
    if status == success:
        status_msg = "OK"
```

```
        else:
            status_msg = "FAILED"
        return status_msg

#first parameter given to program is restore_file
router_dflt_ip = "192.168.1.1"
uname = "root"
passwd = "Password3xample-"
restore_file = sys.argv[1]

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(router_dflt_ip, username=uname, password=passwd)

status = router_cfg(restore_file)
print(status)

ssh.close()
```

```
joram@joram:~/opparitesting$ python testing.py testcfg_10.240.254.cfg
OK
joram@joram:~/opparitesting$
```

*Fig. 17 - In the Python program, "OK" indicates success*

In this case, the program returns "OK" message which translates into "Configuration succesfully updated.". First the file was transferred to the router's /root/ directory, after which the command "*restore testcfg_10.240.254.cfg*" was run. Then, the status message was caught and later used to determine whether the command was succesful or not.

### 4.5 Changing SNMP name

SNMP settings are changed when the new configuration file is put into use. Why couldn't the SNMP name be updated to the configuration file just like other settings? This is because the SNMP name needs to be router's serial number and one doesn't know the serial number yet when the configuration files are created. It would be difficult to put this information into the configuration file, and wreak havoc if serial numbers and files got mixed up.

SNMP name configuration can be found under /etc in settings.snmp file.

```
# cat /etc/settings.snmp
SNMP_ENABLED=1
SNMP_NAME=5322593
SNMP_LOCATION=
SNMP_CONTACT=
```

*Fig. 18 - A snippet of settings inside /etc/settings.snmp*

There is a great command line tool available to edit text files, *sed* (stream line editor). It is a really powerful tool escpecially for text editing and data mining.

```
# sed -i 's/SNMP_NAME=.*/SNMP_NAME=testname/' /etc/settings.snmp
# cat /etc/settings.snmp
SNMP_ENABLED=1
SNMP_NAME=testname
SNMP_LOCATION=
SNMP_CONTACT=
```

*Fig. 19 - sed makes magic happen*

"*sed -i 's/SNMP_NAME=.*/SNMP_NAME=testname/' /etc/settings.snmp*" was run. As can be seen in Figure 19, SNMP name changed to "testname". Now in the real case, the router's serial number will be used instead of ”testname”.

```
import paramiko
import sys

def get_serial():
    cmd = "status -v sys |grep \"Serial Number\" |awk '{print $4}'"
    ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
    outp = ssh_stdout.readlines()
    serial = outp[0].strip()
    return serial

#because there's no output in the command, another command is run to check the new SNMP name
def change_snmp(serial):
    cmd = "sed -i 's/SNMP_NAME=.*/SNMP_NAME=" + str(serial) + "/' /etc/settings.snmp"
```

```
        check = "sed -n 's/SNMP_NAME=//p' /etc/settings.snmp"
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(check)
        outp = ssh_stdout.readlines()
        serialcheck = outp[0].strip()
        if str(serialcheck) == str(serial):
                state = "OK"
        else:
                state = "FAILED"
        return state

router_dflt_ip = "192.168.1.1"
uname = "root"
passwd = "Password3xample-"

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(router_dflt_ip, username=uname, password=passwd)

serial = get_serial()
status = change_snmp(serial)
print(status)

ssh.close()
```

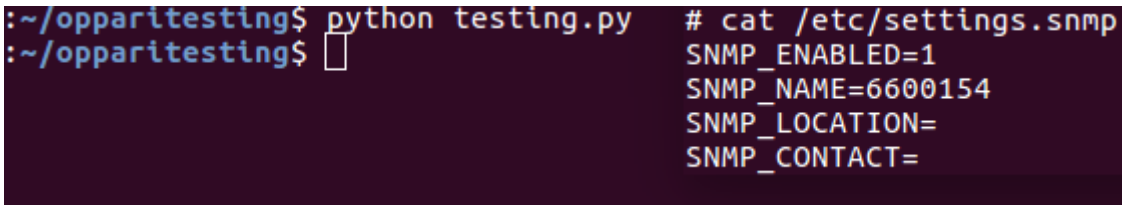


*Fig. 20 - Succesfully changed SNMP name*

As can be seen again in Figure 20, SNMP name has changed. This time the Python program first called get_serial() function to get router's serial number, after which it called change_snmp() function with one argument which was the serial number of the router. Changing SNMP name is not rocket science, but there's always a possibility that something goes wrong. This is the reason why even the simplest change should be verified.

### 4.6 Adding user modules

User modules are third party programs than can be added to routers. User modules are located under /opt directory. User modules are added as .tgz files, which will be decompressed and put into /opt directory. The base structure for this function is similar to restore_cfg() function.

```
import paramiko
import sys

#sftp is used to transfer user module to router
#.tgz is decompressed to /opt
#check is made to make sure the user module exists
#return OK or FAILED
def add_um(user_m, m_name):
    orig = user_m
    dest = "/opt/" + user_m
    cmd = "tar -xzf " + dest + " -C /opt/"
    check = "if [ -d \"/opt/" + m_name + "\" ];then echo OK;else echo NOT;fi"
    sftp = ssh.open_sftp()
    sftp.put(orig, dest)
    ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
    ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(check)
    outp = ssh_stdout.readlines()
    status = outp[0].strip()
    if status == "OK":
        status_msg = "OK"
    else:
        status_msg = "FAILED"
    return status_msg



router_dflt_ip = "192.168.1.1"
uname = "root"
passwd = "Password3xample-"
user_m1 = "pinger.v3.tgz"
user_m1_name = "pinger"
```

```
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(router_dflt_ip, username=uname, password=passwd)

status = add_um(user_m1):
print(status)

ssh.close()
```

```
joram@joram:~/opparitesting$ python testing.py
OK
```

*Fig. 21 - Program returns "OK" status message indicating success*

```
# ls -l /opt/ |grep pinger
drwxrwxr-x    6 root     root            0 Oct 19 15:21 pinger
#
```

*Fig. 22 - Pinger module can be seen under /opt now*

Again, Python program returns "OK" message, so in other words the pinger module has been successfully transferred to the router. This function can be reused as many times as needed to add more user modules.

### 4.7 Changing root password

Having passwords in clear text is always questionable, especially when the root account is concerned. In this case, it was agreed that it is okay as long as only authorized people at NDC have read and write permissions to the program.

To change root's password without user interaction, a combination of *echo* and *chpasswd* commands will be used. The final command is "*echo 'root:<password>' |chpasswd*". If the password actually changed, status message "Password for 'root' changed" is printed to standard output. This can be used later to determine whether root's password was changed or not.

```
import paramiko
```

```
import sys

def change_pw(passwd):
        cmd = "echo 'root:"+ str(passwd) + "' |chpasswd"
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
        outp = ssh_stdout.readlines()
        status = outp[0].strip()
        success = "Password for 'root' changed"
        if status == success:
                status_msg = "OK"
        else:
                status_msg = "FAILED"
        return status_msg

router_dflt_ip = "192.168.1.1"
uname = "root"
passwd = "Password3xample-"
new_passwd = "Str0ngerandl0nger!-"

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(router_dflt_ip, username=uname, password=passwd)

status = change_pwd(new_passwd)
print(status)

ssh.close()
```

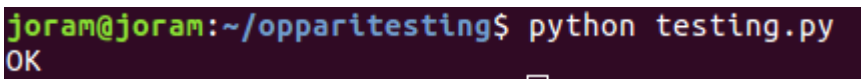


*Fig. 23 - Status message "OK" indicates that the root password was succesfully changed*

The root user's password was succesfully changed. Now the actual configuration part of the process is ready, but this is not the end yet. A backup file has to be downloaded and the excel file needs to be updated.

### 4.8 Getting backup file

Now that configurations are made and verified, a backup file of the current configuration is needed so that it can be stored in NDC's server. The reason why it is needed, is simply because restoring the router’s configuration becomes easy in case if something goes wrong later. This is part of the configuration management. There's command "*backup*" that can be used to create the backup file. Command "*backup*" itself just prints the router's current configuration to standard output, but it can easily be redirected to a file.

```
import paramiko
import sys

#takes one argument, configuration filename
#original filename will be preceded by "bckup"
def get_backup(filename):
        bu_file = "bckup" + filename
        cmd = "backup > " + bu_file
        orig = "/root/" + bufile
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
        sftp = ssh.open_sftp()
        sftp.get(orig, bu_file)

router_dflt_ip = "192.168.1.1"
uname = "root"
passwd = "Password3xample-"
restore_file = sys.argv[1]

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(router_dflt_ip, username=uname, password=passwd)

get_backup(restore_file)

ssh.close()
```

```
joram@joram:~/opparitesting$ python testing.py testcfg_10.240.254.cfg
joram@joram:~/opparitesting$ ls -l
total 104
-rw-rw-r-- 1 joram joram 36240 loka  20 11:42 bckuptestcfg_10.240.254.cfg
-rw-rw-r-- 1 joram joram  8874 loka  19 11:55 fast-lane.py
-rw-rw-r-- 1 joram joram 27688 loka  19 16:20 pinger.v3.tgz
-rw-rw-r-- 1 joram joram 24345 loka  19 13:02 testcfg_10.240.254.cfg
-rw-rw-r-- 1 joram joram  3022 loka  20 11:37 testing.py
joram@joram:~/opparitesting$
```

*Fig. 24 - Backup file can be seen in the directory*

This time, variable bu_file that contains the actual backup filename is used as the destination address of the file. This means the backup file will be saved under the same directory with the Python program.

### 4.9 Catching errors

It is really important to include checks for things that depend on user of the program. For example, does every single file exist in the same directory with the Python program? There are many potential causes of errors. The idea is to get rid of most of them. This makes accidental misuse of the program more difficult.

Firstly, the Python program takes one parameter, an error will be raised if there are no parameters at all or more than one.

```
joram@joram:~/opparitesting$ python testing.py
Traceback (most recent call last):
  File "testing.py", line 102, in <module>
    restore_file = sys.argv[1]
IndexError: list index out of range
joram@joram:~/opparitesting$
```

*Fig. 25 - No parameters to the script caused IndexError*

IndexError indicates that incorrect amount of parameters were given to the program. Also, there are bunch of files defined in the code, so making sure the files really exist in the directory is necessary.

```
joram@joram:~/opparitesting$ python testing.py testcfg_10.240.254.cfg
Traceback (most recent call last):
  File "testing.py", line 111, in <module>
    status = add_um(user_m1, user_m1_name)
  File "testing.py", line 61, in add_um
    sftp.put(orig, dest)
  File "/usr/local/lib/python3.5/dist-packages/paramiko/sftp_client.py", line 719, in put
    file_size = os.stat(localpath).st_size
FileNotFoundError: [Errno 2] No such file or directory: 'pinger.v4.tgz'
joram@joram:~/opparitesting$
```

*Fig. 26 - Missing file caused FileNotFoundError*

For example purposes, the user module "pinger.v3.tgz" was changed to "pinger.v4.tgz" in the code. Because there is no such file, a FileNotFoundError is raised. This can also easily be prevented in the program by checking if the file exists and printing a custom error message.

There are still two probable errors that can easily be spotted by just quickly looking at the code. Both are related to the SSH connection. First, when the SSH authentication happens, incorrect credentials will cause an error.

```
joram@joram:~/opparitesting$ python testing.py testcfg_10.240.254.cfg
Traceback (most recent call last):
  File "testing.py", line 108, in <module>
    ssh.connect(router_dflt_ip, username=uname, password=passwd)
  File "/usr/local/lib/python3.5/dist-packages/paramiko/client.py", line 416, in connect
    look_for_keys, gss_auth, gss_kex, gss_deleg_creds, t.gss_host,
  File "/usr/local/lib/python3.5/dist-packages/paramiko/client.py", line 701, in _auth
    raise saved_exception
  File "/usr/local/lib/python3.5/dist-packages/paramiko/client.py", line 688, in _auth
    self._transport.auth_password(username, password)
  File "/usr/local/lib/python3.5/dist-packages/paramiko/transport.py", line 1378, in auth_password
    return self.auth_handler.wait_for_response(my_event)
  File "/usr/local/lib/python3.5/dist-packages/paramiko/auth_handler.py", line 223, in wait_for_response
    raise e
paramiko.ssh_exception.AuthenticationException: Authentication failed.
joram@joram:~/opparitesting$
```

*Fig. 27 - Wrong credentials caused an error*

The username "root" was changed to "root1" in the code. Because the user doesn't exists, the credentials are deemed incorrect. This caused paramiko's AuthenticationException exception to be raised. Another probable error that can easily be spotted is incorrectly configured network settings. The Python program will keep on trying to connect to the router. Because the network settings are incorrect, the router cannot be found and connection is never established.

```
joram@joram:~/opparitesting$ python testing.py testcfg_10.240.254.cfg
Traceback (most recent call last):
  File "testing.py", line 108, in <module>
    ssh.connect(router_dflt_ip, username=uname, password=passwd)
  File "/usr/local/lib/python3.5/dist-packages/paramiko/client.py", line 331, in connect
    retry_on_signal(lambda: sock.connect(addr))
  File "/usr/local/lib/python3.5/dist-packages/paramiko/util.py", line 276, in retry_on_signal
    return function()
  File "/usr/local/lib/python3.5/dist-packages/paramiko/client.py", line 331, in <lambda>
    retry_on_signal(lambda: sock.connect(addr))
TimeoutError: [Errno 110] Connection timed out
```

*Fig. 28 - TimeoutError occured*

TimeoutError was caused by misconfigured network settings. Anyway, it took a really long time before the exception was raised. Now, it is important to remember that this configuration happens via ethernet cable. Anything more than five seconds indicates that there's something wrong with network settings. Luckily, it is possible to set user defined timeout. Timeout of five seconds should be enough, but it is a good practice to add some room, so timeout is set to 15 seconds.

```
joram@joram:~/opparitesting$ python testing.py testcfg_10.240.254.cfg
Traceback (most recent call last):
  File "testing.py", line 108, in <module>
    ssh.connect(router_dflt_ip, username=uname, password=passwd, timeout=15)
  File "/usr/local/lib/python3.5/dist-packages/paramiko/client.py", line 331, in connect
    retry_on_signal(lambda: sock.connect(addr))
  File "/usr/local/lib/python3.5/dist-packages/paramiko/util.py", line 276, in retry_on_signal
    return function()
  File "/usr/local/lib/python3.5/dist-packages/paramiko/client.py", line 331, in <lambda>
    retry_on_signal(lambda: sock.connect(addr))
socket.timeout: timed out
```

*Fig. 29 - This time timeout was caused by socket*

This time TimeoutError exception was not raised. The error is "socket.timeout". To be able to catch this error, module "socket" needs to be imported. Otherwise, NameError will be raised when catching "socket.timeout". With this knowledge, it is time to strenghten the code and add more logic.

```
import paramiko
import sys
import socket
import os

router_dflt_ip = "192.168.1.1"
uname = "root"
passwd = "Password3xample-"
new_passwd = "Str0ngerandl0nger!-"
```

```
user_m1 = "pinger.v3.tgz"
user_m1_name = "pinger"
user_m2 = "hmpclient.v2.tgz"
user_m2_name = "hmpclient"

#checking that there's one parameter given to the program
try:
        restore_file = sys.argv[1]
except IndexError:
        print("Usage: python autoconfig.py <cfg_file>")
        sys.exit()

#checking if necessary files exist in current working directory
if not os.path.exists(user_m1):
        print("Unable to find user module " + user_m1 + " in current working directory.")
        sys.exit()

if not os.path.exists(user_m2):
        print("Unable to find user module " + user_m2 + " in current working directory.")
        sys.exit()

if not os.path.exists(restore_file):
        print("Unable to find cfg file " + restore_file + " in current working directory.")
        sys.exit()

#timeout is set to 15 seconds
try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(router_dflt_ip, username=uname, password=passwd, timeout=15)
#catching misconfigured network settings
except socket.timeout:
        print("Connection timed out. Check your network settings")
        sys.exit()
#catching incorrect credentials
except paramiko.ssh_exception.AuthenticationException:
        print("Authentication error. Check your credentials")
        sys.exit()
```

```
serial = get_serial()
mac = get_mac()

print("Serial number: " + str(serial))
print("MAC address: " + mac + "\n")
print("Starting configuration daemon...")

print("Sending configuration file...")
while True:
        restore_status = restore_cfg(restore_file)
        if restore_status == "OK":
                break

print("Changing SNMP name...")
while True:
        snmp_status = change_snmp(serial)
        if snmp_status == "OK":
                break

print("Adding user module " + user_m1 + "...")
while True:
        userm_status1 = add_um(user_m1, user_m1_name)
        if userm_status1 == "OK":
                break

print("Adding user module " + user_m2 + "...")
while True:
        userm_status2 = add_um(user_m2, user_m2_name)
        if userm_status2 == "OK":
                break

print("Changing root's password...\n")
while True:
        pw_status = change_pw(new_passwd)
        if pw_status == "OK":
                break

#creating fancy easily spottable summary
print("Verification summary")
```

```
print("---------------------------------")
print("|Configuration file restored: " + restore_status + "|")
print("|SNMP Changed: " + snmp_status + "|")
print("|User module " + user_m1 + ": " + userm_status1 + "|")
print("|User module " + user_m2 + ": " + userm_status2 + "|")
print("|Password changed: " + pw_status + "|")
print("---------------------------------\n")
print("Saving backup configuration file as bckup " + restore_file + "...\n")
get_backup(restore_file)
print("Done. The router can be unplugged now!")

ssh.close()
```

```
joram@joram:~/opparitesting$ python autoconfig.py
Usage: python autoconfig.py <cfg_file>
joram@joram:~/opparitesting$
joram@joram:~/opparitesting$
joram@joram:~/opparitesting$ python autoconfig.py testcfg_10.240.254.cfg
Unable to find user module hmpclient.v2.tgz in current working directory.
joram@joram:~/opparitesting$
joram@joram:~/opparitesting$
joram@joram:~/opparitesting$ python autoconfig.py testcfg_10.240.253.cfg
Unable to find cfg file testcfg_10.240.253.cfg in current working directory.
joram@joram:~/opparitesting$
joram@joram:~/opparitesting$
joram@joram:~/opparitesting$ python autoconfig.py testcfg_10.240.254.cfg
Authentication error. Check your credentials
joram@joram:~/opparitesting$
joram@joram:~/opparitesting$
joram@joram:~/opparitesting$ python autoconfig.py testcfg_10.240.254.cfg
Connection timed out. Check your network settings
joram@joram:~/opparitesting$
```

*Fig. 30 - Exceptions were caught correctly*

As can be seen in Figure 30, error catching works as expected.

```
joram@joram:~/opparitesting$ python autoconfig.py testcfg_10.240.254.cfg
Serial number: 6600154
MAC address: 00:0A:14:84:DF:88

Starting configuration daemon...
Sending configuration file...
Changing SNMP name...
Adding user module pinger.v3.tgz...
Adding user module hmpclient.v2.tgz...
Changing root's password...

Verification summary
---------------------------------
|Configuration file restored: OK |
|SNMP Changed: OK                |
|User module pinger.v3.tgz: OK   |
|User module hmpclient.v2.tgz: OK|
|Password changed: OK            |
---------------------------------

Saving backup configuration file as bckup testcfg_10.240.254.cfg...

Done. The router can be unplugged now!
joram@joram:~/opparitesting$
```

*Fig. 31 - The program ran succesfully*

The router is now configured. While loops are used so that the program will keep on trying until it succeeds in a task. There is a risk that for some unknown reason it never succeeds and will get stuck in an infinite loop, and this could be improved in a future release. For example, the program could try three times and if it doesn't succeed, it returns "FAILED" and skips to the next task.

### 4.10 Updating Excel sheets

Once the router is configured, some information needs to be added to a customer-specific excel file used for documentation and management purposes. Below is an excel template, which is identical to the original one, just without any data.

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | VPN IP | Network Mask | SerialNo | MAC | Model | Date | SWH ok | Zabbix ok | Zabbix ala | Reference | Details |
| 2 | | | | | | | | | | | |
| 3 | | | | | | | | | | | |
| 4 | | | | | | | | | | | |
| 5 | | | | | | | | | | | |
| 6 | | | | | | | | | | | |
| 7 | | | | | | | | | | | |
| 8 | | | | | | | | | | | |
| 9 | | | | | | | | | | | |
| 10 | | | | | | | | | | | |
| 11 | | | | | | | | | | | |
| 12 | | | | | | | | | | | |
| 13 | | | | | | | | | | | |
| 14 | | | | | | | | | | | |
| 15 | | | | | | | | | | | |

*Fig. 32 - An empty excel template*

As it was written earlier, the data which needs to be updated to the excel file is as follows:

- VPN IP address
- Network mask
- Serial Number
- MAC address
- Model
- Date
- Reference

There are also some cells which will be left empty. Those will be updated later if needed. Writing to an excel file with Python is straightforward. In this case, a non-native module *openpyxl* will be used. It is time to write the code and test it with test values. If everything goes according to plan, the code can and will be integrated with the router configuration automation program.

```
import openpyxl

def update_excel():
    #opening excel workbook
    wb = openpyxl.load_workbook(filename = excelfile)

    #opening the first sheet
    sheets = wb.sheetnames
    ws = wb[sheets[0]]

    count = 1
    i = 0
```

```
    #titles are located under these columns, on the same line
    letters = ["A", "B", "C", "D", "E", "F", "J"]
    while 1:
        total = letters[i] + str(count)
        valuecheck = ws[total].value

        #if a cell is empty, write new value
        if not valuecheck:
            if letters[i] == "A":
                ws[total] = fullip
                print("VPN IP updated to excel")
            elif letters[i] == "B":
                ws[total] = mask
                print("Network mask updated to excel")
            elif letters[i] == "C":
                ws[total] = ser
                print("Serial number updated to excel")
            elif letters[i] == "D":
                ws[total] = maci
                print("MAC address updated to excel")
            elif letters[i] == "E":
                ws[total] = model
                print("Model updated to excel")
            elif letters[i] == "F":
                ws[total] = date
                print("Date updated to excel")
            elif letters[i] == "J":
                ws[total] = reference
                print("Reference updated to excel")
                break
            count = 1
            i += 1
        count += 1

    wb.save(filename = excelfile)

fullip = sys.argv[1]
mask = sys.argv[2]
```

```
ser = sys.argv[3]
maci = sys.argv[4]
model = sys.argv[5]
date = sys.argv[6]
reference = sys.argv[7]

excelfile = "customer_router_information.xlsx"

print("Updating excel...\n")
update_excel()
```

```
joram@joram:~/opparitesting$ python fast-lane.py testval1 testval2 testval3 testval4 testval5 testval6 testval7
Updating excel...

VPN IP updated to excel
Network mask updated to excel
Serial number updated to excel
MAC address updated to excel
Model updated to excel
Date updated to excel
Reference updated to excel
```

*Fig. 33 - Test values given as parameters*

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | VPN IP | Network Mask | SerialNo | MAC | Model | Date | SWH ok | Zabbix ok | Zabbix ala | Reference | Details | |
| 2 | testval1 | testval2 | testval3 | testval4 | testval5 | testval6 | | | | testval7 | | |
| 3 | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | |

*Fig. 34 - Data is correctly written to the excel file*

The code itself is very straightforward: locations of the cells with the titles are known; every title is on line one; and only columns differ. For example, SerialNo which represents serial number, can be found at line one and column C (1C). Now when that is known, the code checks if the cell below is empty. If it is empty, the new value is written into it. If it is not empty, it keeps checking the cells below until it finds an empty one and writes into it. In this test, test values were used and they were given as parameters to the program. In the real case when this code is integrated with the router configuration automation program, some of the values can be hardcoded inside the code, some will be provided by the automation program and only a few have to be provided as parameters.

### 4.11 Integration

Integration of these two programs is quite simple. However, a few changes need to be made. First of all, the fewer parameters that has to be provided, the better. Exact model and reference are something that cannot be found inside the router, which is the reason why the information needs to be given as parameters. Luckily, usually neither of the values change within one batch, so re-running the program is fast. Also, earlier the configuration filename was provided as a paremeter, to make it simpler, only the IP address part needs to be provided, for example 10.240.254. This way it can easily be used by update_excel() function and also it is easy to turn it into a filename inside the program.

For clarity, only essential parts are showed below. The actual code in its entirety can be found in Appendix.

```
import paramiko
import sys
import socket
import os
import openpyxl
import time

def get_serial():
    ...

def get_mac():
    ...

def restore_cfg(restore_file):
    ...

def change_snmp(serial):
    ...

def add_um(user_m, m_name):
    ...

def change_pw(passwd):
    ...
```

```
def get_backup(filename):
    ...

def update_excel(ser, maci):
    ...

#checking that there are three parameter given to the program
try:
    model = sys.argv[1]
    reference = sys.argv[2]
    vpnip = sys.argv[3]
except IndexError:
    print("Usage: python autoconfig.py <model> <reference> <first 3 of VPNIP/xxx.xxx.xxx>")
    sys.exit()

router_dflt_ip = "192.168.1.1"
uname = "root"
passwd = "Password3xample-"
new_passwd = "Str0ngerandl0nger!-"
user_m1 = "pinger.v3.tgz"
user_m1_name = "pinger"
user_m2 = "hmpclient.v2.tgz"
user_m2_name = "hmpclient"
restore_file = "testcfg_" + vpnip + ".cfg"
excelfile = "customer_router_information.xlsx"

#information for update_excel()
mask = "255.255.255.0"
fullip = vpnip + ".1"
date = time.strftime("%d/%m/%Y")

serial = get_serial()
mac = get_mac()

#checkin if excel file exists in current directory
if not os.path.exists(excelfile):
    print("Unable to find excel file " + excelfile + " in current working directory.")
    sys.exit()
```

*#----------other error catching code----------*

*#----------configuration and print statements----------*

*update_excel(serial, mac)*

```
joram@joram:~/opparitesting$ python autoconfig.py SR30000 MT2017-10 10.240.254
Serial number: 6600154
MAC address: 00:0A:14:84:DF:88

Starting configuration daemon...
Sending configuration file...
Changing SNMP name...
Adding user module pinger.v3.tgz...
Adding user module hmpclient.v2.tgz...
Changing root's password...

Verification summary
---------------------------------
|Configuration file restored: OK |
|SNMP Changed: OK                |
|User module pinger.v3.tgz: OK   |
|User module hmpclient.v2.tgz: OK|
|Password changed: OK            |
---------------------------------

Saving backup configuration file as bckuptestcfg_10.240.254.cfg...

VPN IP updated to excel
Network mask updated to excel
Serial number updated to excel
MAC address updated to excel
Model updated to excel
Date updated to excel
Reference updated to excel

Done. The router can be unplugged now!
joram@joram:~/opparitesting$
```

*Fig. 35 - Router succesfully configured and data written to the excel file*

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | VPN IP | Network Mask | SerialNo | MAC | Model | Date | SWH ok | Zabbix ok | Zabbix ala | Reference | Details |
| 2 | testval1 | testval2 | testval3 | testval4 | testval5 | testval6 | | | | testval7 | |
| 3 | testval1 | testval2 | testval3 | testval4 | testval5 | testval6 | | | | testval7 | |
| 4 | 10.240.254.1 | 255.255.255.0 | 6600154 | 00:0A:14:84:DF:88 | SR30000 | 23/10/2017 | | | | MT2017-10 | |
| 5 | | | | | | | | | | | |
| 6 | | | | | | | | | | | |

*Fig. 36 - Everything seems to be in place*

Everything works as expected! So, now there's the fully functional code and it can be used in testing and later in production.

# 5 Results

**60** Advantech+BB's SmartFlex LTE routers were configured to make a comparison between manual and automated configuration. 30 of the routers were configured manually, and another 30 by the Python program. Configuration of the routers was conducted in a real production environment, so the results are 100% reliable. A stopwatch was started when the first router was taken out of the box, and stopped when the last configured router was put back into the box. The exact same procedure was applied to the both configuration cases of 30 routers.

## 5.1 Manual configuration

Total time from the beginning to the end in seconds was clocked at **7830** (130 minutes 30 seconds). This means the average configuration time of one router is **261** seconds (4 minutes 21 seconds). Rebooting time of SmartFlex routers is approximately **30** seconds, plugging and unplugging and putting it back to its box takes about **5** seconds. This **~35** seconds could be decreased from the configuration time of one router to get the actual configuration time. During this configuration process, two configuration errors were identified, which had to be fixed as well (wrong SNMP name and mixed MAC addresses in excel).

extras = rebooting, handling router from/to box, plugging/unplugging

**Summary:**

- **Total time of manual configuration:** 2 hours 10 minutes 30 seconds (02:10:30)
- **Total time (minus extras):** ~1 hour 53 minutes and 0 seconds (01:53:00)
- **Average time to configure one router:** 4 minutes 21 seconds (00:04:21)
- **Average time (minus extras):** ~3 minutes 46 seconds (00:03:46)
- **Time taken by extras:** ~17 minutes 30 seconds (00:17:30)
- **Errors:** 2

## 5.2 Automated configuration

Total time from the beginning to the end in seconds was clocked at **1533** (25 minutes 33 seconds). This means the average configuration time of one router is **51.1** seconds. The same extra time that comes from rebooting, plugging etc. applies here. So the actual time the Python program runs can be calculated by decreasing **~35** seconds from one router's configuration time.

extras = rebooting, handling router from/to box, plugging/unplugging

**Summary:**

- **Total time of automated configuration:** 25 minutes 33 seconds (00:25:33)
- **Total time (minus extras):** ~8 minutes and 3 seconds (00:08:03)
- **Average time to configure one router:** 51.1 seconds (00:00:51)
- **Average time (minus extras):** ~16 seconds (00:00:16)
- **Time taken by extras:** ~17 minutes 30 seconds (00:17:30)
- **Errors:** 0

### 5.3 Comparison of results

Configuration of 30 SmartFlex routers using the Python program saved **01:44:57** hours. In other words the automated method took only **~19.5%** of the time of manual configuration.

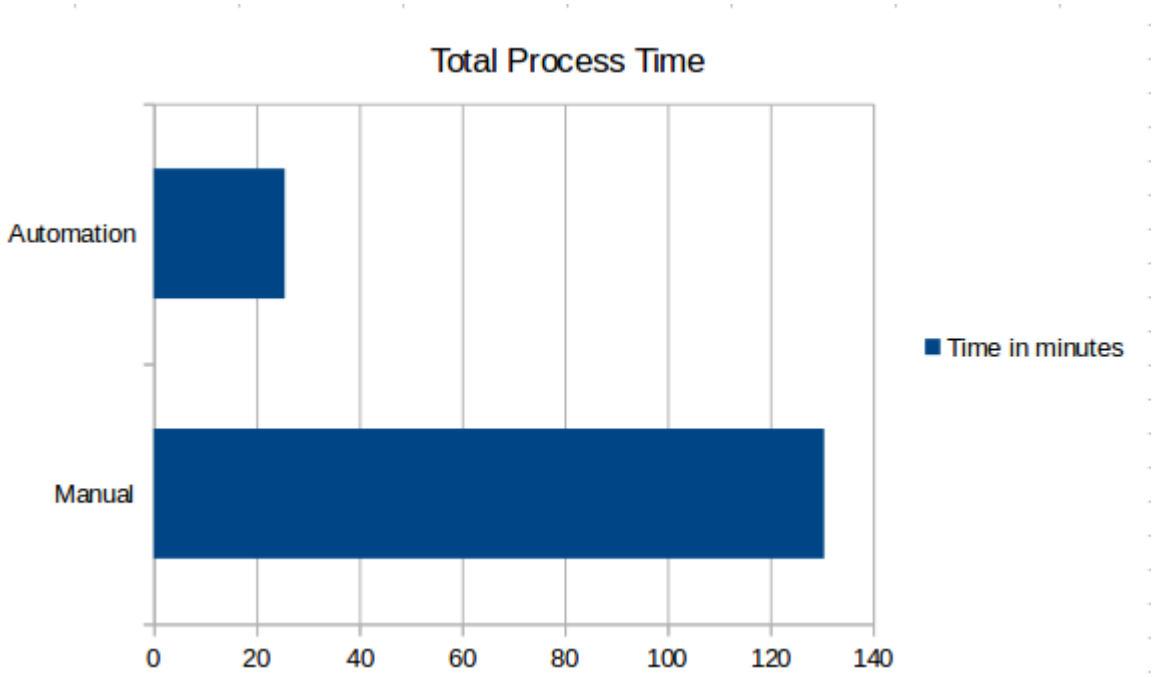


*Fig. 37 - Lots of time can be saved using the Python program*

Now if the rebooting time, router handling time from/to box and plugging/unplugging are left out, the actual configuration with the automation program takes only **~7%** of the time of manual configuration, as can be seen in Fig. 38.

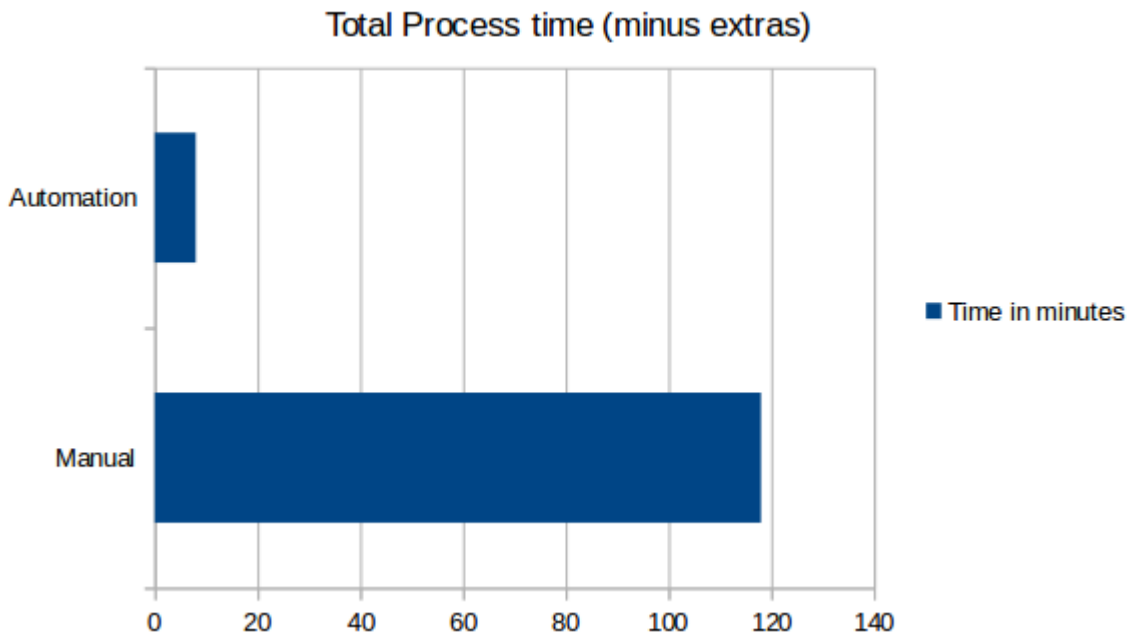


*Fig. 38 - The difference in actual configuration speed is huge*

**60** routers were configured for this test case. A realistic estimation for a year is **3000-4000** routers. It's time to calculate the estimation of manual and automated configuration times for that batch, based on the results above.

**Average configuration time of 1 router (Manual):** 261 seconds
**Average configuration time of 1 router (Automated):** 51 seconds

**Estimated configuration time of 4000 routers (Manual):** 261 seconds * 4000 = 1 044 000 seconds (~12,1 days)
**Estimated configuration time of 4000 routers (Automated):** 51 seconds * 4000 = 204 000 seconds (~2,4 days)

If there was just one person configuring routers, manually it would take over twelve days for the person to finish the configuration process, supposing that the person keeps configuring 24 hours a day, seven days a week. Superman probably doesn't exist, so it's time to make a new estimation. Eight hours is a more realistic working time estimation for a day. It means the configuration time needs to be multiplied by three.

**Estimated configuration time of 4000 routers (Manual):** ~36.3 days
**Estimated configuration time of 4000 routers (Automated):** ~7,1 days

The manual configuration of **4000** routers would take approximately **36 days** (by one person), while the same process using the autmation program would take only around **7 days**. In other words, that would save **29 working days a year**, which is more than a month's salary. Supposing the person's salary is **3500 euros** a month, NDC would save about **4600 euros** a year if the value of a working day was calculated according to the person's salary. In reality though, the work itself can be more valuable by a big margin. Especially considering how much can be done in 29 working days. So the real value in saved time is even bigger than described above.

Additionally, one can only wonder how many misconfigurations could be made when configuring 4000 routers manually. It happened twice with the batch of 30. Based on this, ~267 misconfigurations could be calculated for a batch of 4000 routers. This number doesn't even include the possible imperceptible misconfigurations, which may be noticed later. Additionally, configuring routers manually is tedious, but when using the automation program, one doesn't have to do those many tasks (clicking, browsing, typing) because the program manages them, just run the program. Still most importantly, one doesn't really have to think during the process. Thinking takes a lot of energy and a tired brain is susceptible to make mistakes. Only thing one needs to remember when running the program is to check the VPN IP address which is given as a command line parameter and thus is easy and convenient to change before re-running the program.

### 5.4 Comments from the CEO of NDC Networks, Markus Ahonen

*The automation tools described in this thesis have been taken into active production use at NDC, and they have been used to configure over NNN routers so far.*

*NDC is expecting to deliver over 5000 routers during 2018, so the calculable savings in direct effort are significant – especially as handling this amount of routers would require a new part-time hire, leading to additional costs upward of 20000€/year.*

*Additionally, the tool reduces human error, which can easily cause 10x costs if routers have to be serviced in the field.*

*However, the most important aspect of the automation tool is that it allows NDC technical staff to focus on tasks that utilize their key skills (networking design and problem solving), rather than tedious configuration tasks that reduce job satisfaction and increase the likelihood of employee churn. More specifically, it is likely that no amount of financial compensation will keep a skilled expert motivated or committed to NDC if 20% of their working time is used for repetitive tasks. It takes up to 18 months for even a seasoned expert to learn and master NDC's customer segment needs and specifics – so the longevity of employment is critical to NDC's ability to provide its technical consulting and support services to its customers. Due to this, a likely future development for the automation tool is to create a*

*user interface that allows warehouse staff to complete the configuration tasks without assistance from the technical team. This further reduces the repetitive tasks from key technical experts, and removes them as a dependency from NDC's supply & logistics operations.*

# 6 Future development

There are things that could be improved, both inside and outside the Python program. When configuring the routers with the program, rebooting time is longer than the actual configuration time. This is something that only the manufacturer can change though.

Configuration speed is not a problem anymore. Anyway, some more checks could be made by the program. Currently, it checks if there are exactly three parameters provided when run and whether all the needed files exist within the same directory. Even if the credentials are wrong or network settings are misconfigured, the program helps to point out the problem. Still, currently the parameter values are not checked. The provided parameters should be verified to match desired criteria.

Taking it even further, it could be a good idea to get rid of parameters. A list of possible hardware models could be made. For each model, a unique feature should be found and based on that the program would choose the correct model from a list. There could also be a list of router configuration filenames in numeric order. The program could then read the first line in the list, configure the respective router and remove the first line from the list, so the list would be empty after configuring the last router.

Also, currently when something is transferred to a router by sftp, the program verifies the transfer by checking whether the file exists in the router. Downside is, it doesn’t indicate if the file was succesfully transferred. Integrity of the transferred file should be checked. This could be accomplished by comparing md5 hashes of the origin and destination.

# 7 Conclusion

Manual configuration method, at least in this particular case is slow and inefficient. Fortunately, better methods exist. Web scraping and SSH configuration were explained in more details, because they were easily applicable to this particular case. Automated configuration over SSH was chosen as the used method in this thesis due to its reliability compared to Web scraping techniques.

This thesis produced a configuration tool written in Python3 that first establishes a connection between a configuring system and a router, after which it executes functions that make the actual configuration. When the configuration part is done, it updates a customer's excel file with desired information. The configuration tool makes different checks to verify the success of configuration. Future development part of the thesis proposes improvements on the configuration tool.

Results of comparison between automated and manual configuration were introduced in chapter 5. The results clearly indicate how much more efficient the automated method is compared to manual configuration. The Python tool's financial value is also made very clear by examples and in comments from NDC Networks' CEO Markus Ahonen.

# 8 References


**Advantech B+B**. (read 27.10.2017). SmartFlex LTE & LAN Router (Spectre V3 LTE – ERT). Retrieved from http://advantech-bb.com/new-products/Products/Cellular-Wireless/cellular-routers/smartflex-lte-lan-router-spectre-v3-lte-ert/

**Advantech B+B**. (read 27.10.2017). SmartStart LTE Router. Retrieved from http://advantech-bb.com/newproducts/Products/CellularWireless/cellular-routers/smartstart-lte-router/

**Cisco**. (2013). Configuration Guide; Chapter: Using the Command-Line Interface. Retrieved from https://www.cisco.com/c/en/us/td/docs/ios/12_2/configfun/configuration/guide/ffun_c/fcf001.html

**Cisco**. (2014). Cisco IOS Scripting with TCL Configuration Guide, Cisco IOS Release 15M&T. Retrieved from https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ios_tcl/configuration/15-mt/ios-tcl-15-mt-book/nm-script-tcl.html

**Dart, S**. 1991. Concepts in Configuration Management Systems. Pages 1-2. Retrieved from ftp://itin.sei.cmu.edu/pub/case-env/config_mgt/papers/cm_concepts.pdf

**Enck, McDaniel, Sen, Psebos, sspoerel, albert, sanjay, aiello.** (2007). Configuration Management at Massive Scale: System Design and Experience. Retrieved from https://www.usenix.org/legacy/event/usenix07/tech/full_papers/enck/enck_html/

**Ernst, T. & Lach, H-Y.** (2007). Network Mobility Support Terminology. Retrieved from https://tools.ietf.org/pdf/rfc4885.pdf

**Finder**. (2017). NDC Networks Oy; Taloustiedot. Retrieved from https://www.finder.fi/Internet-palveluja/NDC+Networks+Oy/Espoo/yhteystiedot/159585

**Glez-Peña, D., Lourenço, A., López-Fernández H., Reboiro-Jato M., Fdez-Riverola F**.; Web scraping technologies in an API world, Briefings in Bioinformatics, Volume 15, Issue 5, 1 September 2014, Pages 788–797.
Retrieved from
https://doi.org/10.1093/bib/bbt026

**Griffioen, R., de Haan, J., Willenborg L.** (2014). Collecting clothing data from the Internet. Pages 14-15.
Retrieved from
http://www.unece.org.net4all.ch/fileadmin/DAM/stats/documents/ece/ces/ge.22/2014/UNECE-ILO_2014_Griffioen_deHaan_Willenborg.pdf

**Haddaway, N**. 2015. The Use of Web-scraping Software in Searching for Grey Literature. Grey Journal. 11. pages 186-190.
Retrieved from
https://www.researchgate.net/publication/282658358_The_Use_of_Webscraping_Software_in_Searching_for_Grey_Literature

**Heydon, A. & Nojark, M**. (1999). Mercator: A scalable, extensible Web crawler.
Retrieved from
http://www.bagualu.net/linux/crawler.pdf

**IDC**. (2017). IDC's Worldwide Quarterly Ethernet Switch and Router Trackers Show Steady Growth for Q1 2017; Results Bode Well for Year Ahead.
Retrieved from
https://www.idc.com/getdoc.jsp?containerId=prUS42757317

**IEEE.** (1990). Standard Glossary of Software Engineering Terminology. Pages 20-21.
Retrieved from
http://www.mit.jyu.fi/ope/kurssit/TIES462/Materiaalit/IEEE_SoftwareEngGlossary.pdf

**Khan, I. 2014**. Cisco Vs Huawei CLI 1-2 (Basic Commands 2).
Retrieved from
http://www.networksheaven.com/wp-content/gallery/cisco-vs-huawei-cli-commands/ciscovshuaweicli-basiccommands2.jpg

**Lawson, R.** 2015. Web Scraping With Python; Scrape data from any website with the power of Python. Pages 31-32.
Retrieved from

https://marcell.memoryoftheworld.org/Richard%20Lawson/Web%20Scraping%20With%20Python%20(2685)/Web%20Scraping%20With%20Python%20-%20Richard%20Lawson.pdf

**Sarma, A., Noroozi, Z., van der Hoek, A.** 2003. Palantír: Raising Awareness among Configuration Management Workspaces. Page 1.
Retrieved from
https://pdfs.semanticscholar.org/229e/8f8747999555644dc3a1cb1eb840eaf66c9b.pdf

# 9 Appendix

The source code of autoconfig.py

```
import paramiko
import sys
import socket
import os
import time
import openpyxl

def get_serial():
    cmd = "status -v sys |grep \"Serial Number\" |awk '{print $4}'"
    ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
    outp = ssh_stdout.readlines()
    serial = outp[0].strip()
    return serial

def get_mac():
    cmd = "ifconfig eth0 |grep \"HWaddr\" |awk '{print $5}'"
    ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
    outp = ssh_stdout.readlines()
    mac = outp[0].strip()
    return mac

def restore_cfg(restore_file):
    orig = restore_file
    dest = "/root/" + restore_file
    cmd = "restore " + dest
    sftp = ssh.open_sftp()
    sftp.put(orig, dest)
    ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
    success = "Configuration successfully updated."
    outp = ssh_stdout.readlines()
    status = outp[0].strip()
    if status == success:
        status_msg = "OK"
    else:
```

```
            status_msg = "FAILED"
        return status_msg

def change_snmp(serial):
        cmd = "sed -i 's/SNMP_NAME=.*/SNMP_NAME=" + str(serial) + "/' /etc/settings.snmp"
        check = "sed -n 's/SNMP_NAME=//p' /etc/settings.snmp"
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(check)
        outp = ssh_stdout.readlines()
        serialcheck = outp[0].strip()
        if str(serialcheck) == str(serial):
            state = "OK"
        else:
            state = "FAILED"
        return state

def add_um(user_m, m_name):
        orig = user_m
        dest = "/opt/" + user_m
        cmd = "tar -xzf " + dest + " -C /opt/"
        check = "if [ -d \"/opt/" + m_name + "\" ];then echo OK;else echo NOT;fi"
        sftp = ssh.open_sftp()
        sftp.put(orig, dest)
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(check)
        outp = ssh_stdout.readlines()
        status = outp[0].strip()
        if status == "OK":
            status_msg = "OK"
        else:
            status_msg = "FAILED"
        return status_msg

def change_pw(passwd):
        cmd = "echo 'root:"+ str(passwd) + "' |chpasswd"
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
        outp = ssh_stdout.readlines()
        status = outp[0].strip()
        success = "Password for 'root' changed"
```

```
    if status == success:
        status_msg = "OK"
    else:
        status_msg = "FAILED"
    return status_msg

def get_backup(filename):
    bu_file = "bckup" + filename
    cmd = "backup > " + bu_file
    orig = "/root/" + bu_file
    ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(cmd)
    sftp = ssh.open_sftp()
    sftp.get(orig, bu_file)

def update_excel(ser, maci):
    #opening excel workbook
    wb = openpyxl.load_workbook(filename = excelfile)

    #opening the first sheet
    sheets = wb.sheetnames
    ws = wb[sheets[0]]

    count = 1
    i = 0

    letters = ["A", "B", "C", "D", "E", "F", "J"]
    while 1:
        total = letters[i] + str(count)
        valuecheck = ws[total].value
        if not valuecheck:
            if letters[i] == "A":
                ws[total] = fullip
                print("VPN IP updated to excel")
            elif letters[i] == "B":
                ws[total] = mask
                print("Network mask updated to excel")
            elif letters[i] == "C":
                ws[total] = ser
                print("Serial number updated to excel")
```

```
                elif letters[i] == "D":
                        ws[total] = maci
                        print("MAC address updated to excel")
                elif letters[i] == "E":
                        ws[total] = model
                        print("Model updated to excel")
                elif letters[i] == "F":
                        ws[total] = date
                        print("Date updated to excel")
                elif letters[i] == "J":
                        ws[total] = reference
                        print("Reference updated to excel")
                        break
                count = 1
                i += 1
            count += 1

        wb.save(filename = excelfile)

try:
        model = sys.argv[1]
        reference = sys.argv[2]
        vpnip = sys.argv[3]
except IndexError:
        print("Usage: python autoconfig.py <model> <reference> <first 3 of VPNIP/xxx.xxx.xxx>")
        sys.exit()

router_dflt_ip = "192.168.9.1"
uname = "root"
passwd = "<password>"
new_passwd = "<password>"
user_m1 = "pinger.v3.tgz"
user_m1_name = "pinger"
user_m2 = "hmpclient.v2.tgz"
user_m2_name = "hmpclient"
date = time.strftime("%d/%m/%Y")
excelfile = "customer_router_information.xlsx"
mask = "255.255.255.0"
restore_file = "testcfg_" + vpnip + ".cfg"
```

```
fullip = vpnip + ".1"

#checking if necessary files exist in current working directory
if not os.path.exists(user_m1):
    print("Unable to find user module " + user_m1 + " in current working directory.")
    sys.exit()

if not os.path.exists(user_m2):
    print("Unable to find user module " + user_m2 + " in current working directory.")
    sys.exit()

if not os.path.exists(restore_file):
    print("Unable to find cfg file " + restore_file + " in current working directory.")
    sys.exit()

if not os.path.exists(excelfile):
    print("Unable to find excel file " + excelfile + " in current working directory.")
    sys.exit()

try:
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(router_dflt_ip, username=uname, password=passwd, timeout=15)
#catching misconfigured network settings
except socket.timeout:
    print("Connection timed out. Check your network settings")
    sys.exit()
#catching incorrect credentials
except paramiko.ssh_exception.AuthenticationException:
    print("Authentication error. Check your credentials")
    sys.exit()

serial = get_serial()
mac = get_mac()

print("Serial number: " + str(serial))
print("MAC address: " + mac + "\n")
print("Starting configuration daemon...")
```

```
print("Sending configuration file...")
while True:
        restore_status = restore_cfg(restore_file)
        if restore_status == "OK":
                break

print("Changing SNMP name...")
while True:
        snmp_status = change_snmp(serial)
        if snmp_status == "OK":
                break

print("Adding user module " + user_m1 + "...")
while True:
        userm_status1 = add_um(user_m1, user_m1_name)
        if userm_status1 == "OK":
                break

print("Adding user module " + user_m2 + "...")
while True:
        userm_status2 = add_um(user_m2, user_m2_name)
        if userm_status2 == "OK":
                break

print("Changing root's password...\n")
while True:
        pw_status = change_pw(new_passwd)
        if pw_status == "OK":
                break

print("Verification summary")
print("---------------------------------")
print("|Configuration file restored: " + restore_status + " |")
print("|SNMP Changed: " + snmp_status + "               |")
print("|User module " + user_m1 + ": " + userm_status1 + "   |")
print("|User module " + user_m2 + ": " + userm_status2 + "|")
print("|Password changed: " + pw_status + "           |")
print("---------------------------------\n")
print("Saving backup configuration file as bckup" + restore_file + "...\n")
```

*get_backup(restore_file)*
*update_excel(serial, mac)*
*print("\nDone. The router can be unplugged now!")*

*ssh.close()*

Installing *openpyxl* and *paramiko* with pip

On a command line, type the following two commands

*pip3 install paramiko*

*pip3 install openpyxl*