Automatic network configuration in virtualized environment using GNS3

Rodrigo Emiliano*, Mário Antunes*†
*School of Technology and Management, Polytechnic Institute of Leiria, Portugal
2130975@my.ipleiria.pt, mario.antunes@ipleiria.pt
†Center for Research in Advanced Computing Systems, University of Porto, Portugal
mantunes@dcc.fc.up.pt

***Abstract*—Computer networking is a central topic in computer science courses curricula offered by higher education institutions. Network virtualization and simulation tools, like GNS3, allows students and practitioners to test real world networking configuration scenarios and to configure complex network scenarios by configuring virtualized equipments, such as routers and switches, through each one's virtual console.**

The configuration of advanced network topics in GNS3 requires that students have to apply basic and very repetitive IP configuration tasks in all network equipments. As the network topology grows, so does the amount of network equipments to be configured, which may lead to logical configuration errors.

In this paper we propose an extension for GNS3 network virtualizer, to automatically generate a valid configuration of all the network equipments in a GNS3 scenario. Our implementation is able to automatically produce an initial IP and routing configuration of all the Cisco virtual equipments by using the GNS3 specification files. We tested this extension against a set of networked scenarios which proved the robustness, readiness and speedup of the overall configuration tasks. In a learning environment, this feature may save time for all networking practitioners, both beginners or advanced, who aim to configure and test network topologies, since it automatically produces a valid and operational configuration for all the equipments designed in a GNS3 environment.

***Keywords*-Automatic configuration, GNS3, virtualization, Cisco IOS, virtual lab classroom.**

I. Introduction

Computer networking is actually part of all the relevant computer science courses curricula that are running in higher education institutions worldwide [1]. In its last revision, ACM curriculum guidelines report for undergraduate degree programs in computer science [2] included Networking and Communication (NC) as a knowledge area in computer science curricula, replacing Net-Centric Computing curricula topic that was introduced in the revision of 2001. This measure is explainable by the growing number of computing activities and applications that are ubiquitous and strongly depend on the correct operation of the underlying network [2].

Learning and teaching networking topics in a classroom requires an effective access to network equipments (e.g. routers and switches) so the students can have an hands-on experience with the configuration and troubleshooting of complex and heterogeneous scenarios. Such approaches have however some weaknesses: i) setting up a computer networking lab may be expensive; ii) equipments become obsolete and may malfunction; iii) students are not able to have these equipments at home to practice, which may lead to a learning productivity loss [1].

The use of simulators and hypervisors for network equipments have emerged in the last years [3]. Networking virtualizers, like GNS3, are mostly user-friendly and allow users to configure a network device in a virtual machine running the same operating system as the real network device. Network simulators (e.g Cisco Packet Tracer) are also graphical but more limited since they usually provide a subset of operating system commands and configuration tasks.

Basic networking topology configuration has to follow strict operations. First we need to define a logical IP configuration and apply it to all connected interfaces cards of all the equipments. Then we have to configure a routing protocol, such as RIP or OSPF. Route announcements and node neighborhood establishment have to be done manually and must be consistent with the logical IP configuration previously made. Finally, there is also a set of general operations that have to be done, like hostname and remote access levels definition, enabling logging and access lists definition. Depending on the amount of interface cards and equipments, these essential and basic configuration tasks can be very tedious, repetitive and prone to human mistakes.

For a beginner practitioner these repeated tasks are important to explore the command line and to gain experience on configuring a network device. However, for more experienced ones, who are able to hands-on in more advanced topics, these initial configuration tasks are not challenging and, depending on the amount of equipments involved, may be very time consuming.

In this paper we propose an extension for GNS3 to automatically generate the configuration of all the Cisco network equipments that are part of a virtual topology. We take advantage of the topology setup configuration files produced by GNS3 to automatically generate the nodes (e.g. routers and switches) configuration files, without the need to access their corresponding consoles. Our implementation is also able to include customized settings, like the routing protocol to be used and which baseline configuration should be applied. This extension for GNS3 allows beginner practitioners to produce valid configuration files that can be further analysed, replicated and even applied to real network equipments. For more experienced students or practitioners this extension allows to speedup the overall network configuration task in large and complex network topologies, since they may overtake some initial and repetitive IP configuration steps.

The rest of the paper is organized as follows. We start in Section II by describing and comparing the most relevant automatic configuration solutions for networks. We then proceed in Section III by detailing our proposed approach. In Section IV we explain the major issues regarding the development. In Section V we present and analyse the results obtained with the use of our application in classroom lab scenarios. Finally, in Section VI we present the most relevant conclusions and delineate some directions for future work.

II. Related Work

Self-study, certification, proficiency and long-life learning in networking skills can be achieved by using physical equipments, but it is fundamentally done by using specific and dedicated simulators and hypervisors [3].

There is an evident difference between virtualization of network devices [4] and simulation [5]. While simulators can be a helping hand in the initial learning stages, they become limited when the network topologies get more complex, both in size and in the protocols involved, since only a subset of features are usually available. Some examples of simulators are Cisco Packet Tracer or Huawei eNSP simulator [6] [7] [8]. A different approach is provided by virtualized applications that are able to manage a set of virtual machines running the same operating system kernel version of the real network device. GNS3 [9] [10] is one of the most commonly used application to virtualize Cisco networks. GNS3 hypervisor, `dynamips`, manages a topology of internetworked Cisco devices and allows the access to a virtual console port and consequently to the Cisco IOS command line. Virtualized network topology settings can be changed according to memory and CPU configuration in the host system. Virtualized devices in GNS3 can also interact with real routers and networks connected to the host system, providing full integration between virtualized and real world scenarios and thus extending the whole network topology [10]. Configuration files are plain text and identical in both real and virtual routers managed by GNS3. So, GNS3 allows network practitioners to learn networking technologies and to test real world networking configurations without the need of having physical equipments.

Besides network virtualizers and simulators, there is a wide range of applications that help to automatically produce the configuration of network equipments. Some examples are Netomata Config Generator (`http://www.netomata.com`), Gen-IT (`http://gen-it.net`), AutoNetKit (`http://autonetkit.org`) and Solarwinds Network Config Generator (`http://www.solarwinds.com`).

These applications use different approaches to automate the generation of configuration files. Solarwinds Network Config Generator only permits *one router at a time* configuration. It allows custom templates that can be based on Solarwinds' *Thwack* online community templates.

Table I
Applications tested for network automatic configuration.

	Netomata Config Gen.	Solarwinds	GEN-IT	AutoNetKit
License	GPL	GPL	Windows	GPL
Multi-vendor	Yes	No	Yes	Yes
Support	No	Yes	No	Yes
Effectiveness	Yes	Yes	No	Yes
GUI	No	Yes	Yes	Yes

Gen-IT is available only for Windows, is proprietary and uses spreadsheets and templates for mass configuration. AutoNetKit uses a graphical user interface for network design and virtualization. It has network flow simulation and analysis functions through a web interface, which allows the visualization of the designed network's nodes and their interactions [11]. Netomata Config Generator uses a command-line approach for network configuration. It is based on python specification files to determine the nodes and their interaction.

We have installed, tested and evaluated these applications, since they are in some way, in line with our proposal to automatically generate a network configuration. Table I compares the applications in some essential features.

While existing applications for automatic configuration have interesting features, there are still many challenges and improvements to be made. Some applications did not have any documentation or user and development support. Some of them did not create the bulk configuration to be loaded into the devices, while others only allow to configure one equipment at a time. An important issue is related with multi-vendor support, since almost all the applications are only available to Cisco operating system (IOS) or did not have any clarification on how can be implemented on a multi-vendor network topology.

After analysing the existing approaches for automatic configuration and also identifying the benefits of using GNS3 network virtualizer, we present in next section the proposed approach for our work, which consists on an application that can be integrated on GNS3, to automatically generate initial configuration files.

III. Proposed Approach

In this section we propose an extension to GNS3 network virtualizer that can automatically generate the initial configuration files of all the routers involved in a network scenario. GNS3 graphical interface allows the user to easily design the topology, which produces a text specification file with relevant information for each object. Our application reads this configuration file and generates the Cisco IOS configuration file related to each device. The application will produce a valid and working configuration files set, based on some general settings, like the way IP addresses are chosen and which routing protocol will be used.

Figure 1 illustrates the proposed architecture. After drawing the topology with GNS3, the resulting `JSON` project file (`.gns3`) has the network topology specification, which includes all the nodes specification and the relations between

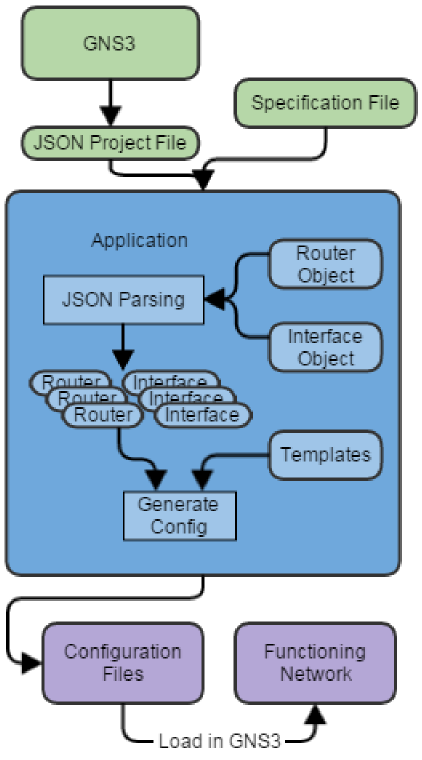


Figure 1. Proposed architecture.

them. Our application, a set of scripts developed in Perl, receives two input files as parameters: the `.gns3` file and a specification file with settings related with the way configuration has to be created. After parsing the GNS3 file, the script will automatically generate the initial configuration files of all the network devices, in such a way that the logical topology and IP configuration are valid and operational. Those configuration files will be generated using predefined templates, in which we define the commands required for each function, like setting IP in an interface card or configuring a specific routing protocol. After the file generation stage has been completed and since GNS3 stores the configuration files for each node in a network project, the newly generated configuration files produced by our script will replace the initial (and empty) ones. Finally, since we start the Cisco IOS virtual machines through GNS3, the newly created configuration files will be loaded for each corresponding device, represented by a separated virtual machine.

The way our script uses the correct Cisco IOS commands for each configuration line is through the use of *templates*. For example, to give an IP configuration on an interface card, one must have to use the Cisco IOS command `ip address a.b.c.d x.y.z.w`, being "`a.b.c.d`" and "`x.y.z.w`" the IP address and subnet mask respectively.

There is a template for each specific function. For example, there is a template dedicated to each routing protocol configuration, another one for interface card configuration, and so forth. Each command in a template may have one or more masks to be replaced by values generated by the script or provided by the user. Figure 2 illustrates a template used to configure a interface card.

```
interface __INTERFACE__
ip address __ADDRESS__ __MASK__
description __DESCRIPTION__
no shutdown
```

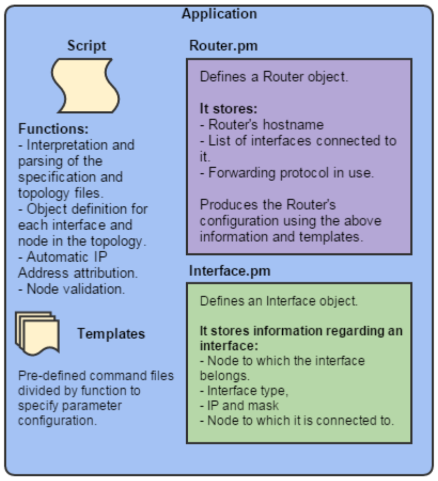
Figure 2. An example of a template.

The masks `__INTERFACE__`, `__ADDRESS__`, `__MASK__` and `__DESCRIPTION__` will be replaced by the script with the values corresponding to an interface object belonging to the router being configured, namely the ip address, subnet mask and description.

In the next section we describe the major ideas regarding the development of our application.

IV. Development

Our application was developed on Perl, a high level all-purpose programming language that excels on the reading and processing of data. We took advantage on intrinsic Perl Object Oriented Programing (OOP) features, since there is a lot of processing and storage of data. Figure 3 illustrates how the application is organized and which developed modules are being used, along with some relevant features.



Functions:
- Interpretation and parsing of the specification and topology files.
- Object definition for each interface and node in the topology.
- Automatic IP Address attribution.
- Node validation.

Pre-defined command files divided by function to specify parameter configuration.

Defines a Router object.

It stores:
- Router's hostname
- List of interfaces connected to it.
- Forwarding protocol in use.

Produces the Router's configuration using the above information and templates.

Defines an Interface object.

It stores information regarding an interface:
- Node to which the interface belongs.
- Interface type,
- IP and mask
- Node to which it is connected to.

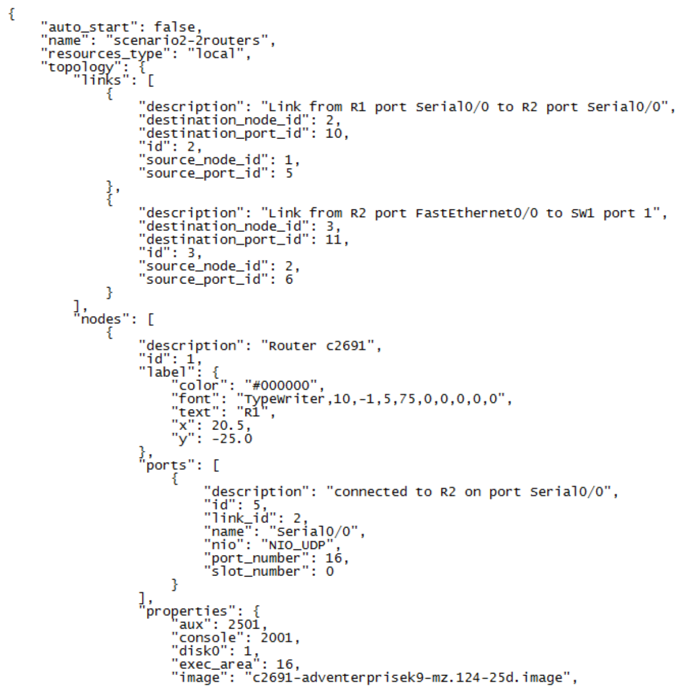
Figure 3. Main components of the application.

The application is modular and organized in the following main modules: *script*, *router* and *interface*. The script is the starting point for processing, where all the functions are invoked, like parsing, data storage and config generation. The Router Perl module (`Router.pm`) defines the settings for a router object and stores all the information regarding to it, such as the hostname, existing interfaces and forwarding protocol in use. It also combines these settings to further create the resulting configuration file for each router. The Interface Perl module (`Interface.pm`) stores all the information regarding the existing connections between the routers in the network topology and which interface cards are involved.

In the development phase we took the following major assumptions: i) only routers and switches Cisco IOS are supported by GNS3 and thus allowed in working network topologies; ii) for the sake of simplicity, switches configurations are not considered, that is automatic configuration will only apply to routers devices; iii) for `serial` links, `clock rate` command is always configured in both sides of the link.

A. *JSON parser and data storage*

This function collects the necessary information about the network topology and parses the JSON specification file, produced by GNS3. To do this, it uses a specific JSON library (JSON Perl module, available in `CPAN`) to decode the file and to load its contents to a local data structure. By using `Data::Dumper` function it is possible to print the data structure. Figure 4 illustrates how data is saved in `.gns3` specification file, in the JSON format.



```
{
    "auto_start": false,
    "name": "scenario2-2routers",
    "resources_type": "local",
    "topology": {
        "links": [
            {
                "description": "Link from R1 port Serial0/0 to R2 port Serial0/0",
                "destination_node_id": 2,
                "destination_port_id": 10,
                "id": 2,
                "source_node_id": 1,
                "source_port_id": 5
            },
            {
                "description": "Link from R2 port FastEthernet0/0 to SW1 port 1",
                "destination_node_id": 3,
                "destination_port_id": 11,
                "id": 3,
                "source_node_id": 2,
                "source_port_id": 6
            }
        ],
        "nodes": [
            {
                "description": "Router c2691",
                "id": 1,
                "label": {
                    "color": "#000000",
                    "font": "TypeWriter,10,-1,5,75,0,0,0,0,0",
                    "text": "R1",
                    "x": 20.5,
                    "y": -25.0
                },
                "ports": [
                    {
                        "description": "connected to R2 on port Serial0/0",
                        "id": 5,
                        "link_id": 2,
                        "name": "Serial0/0",
                        "nio": "NIO_UDP",
                        "port_number": 16,
                        "slot_number": 0
                    }
                ],
                "properties": {
                    "aux": 2501,
                    "console": 2001,
                    "disk0": 1,
                    "exec_area": 16,
                    "image": "c2691-adventerprisek9-mz.124-25d.image",
```

Figure 4. GNS3 JSON file format.

By iterating the JSON specification file, all the data regarding devices and links in the network topology is collected and stored. This information is crucial to know how the nodes are interconnected, with which neighbour router and by which network interface cards. Next, we assign automatically a different IP network to each one of the interface cards configured on each router. At the moment we are using the IP range 192.168.0.0/16, which has approximately 2^{16} different IP networks. We assign a /24 network address that is incremented depending on the number of links in the topology. For instance, two network interface cards in a router will be configured in the IP networks 192.168.0.0/24 and 192.168.1.0/24, with the addresses 192.168.0.1 and 192.168.1.1 respectively. Depending on the network topology, new IP addresses are being allocated sequentially in these IP networks, for all the devices in the same broadcast domain. Information about router connections and IP networks are stored in a Interface object for each router interface. Each interface object has a variable that specifies the router with which this Interface is connected to, that has to be in the same IP network.

After storing all the data regarding the links in the JSON file, the next step is to store the information about the devices. Depending on the device type, different settings may be collected. For routers, the only device type available at the moment, the objects store the hostname, the router type and the interfaces that belong to it, which were created earlier.

B. *Automatic configuration generator*

After storing all the information related to links and devices, using `Interface.pm` and `Router.pm` Perl modules, we proceed to create automatically the configuration files. This is done by calling the a specific method (`createAllConfigurations()`) defined in `Router.pm` Perl Module. Method `createConfiguration()` will be executed for each instantiated router object, in order to produce the corresponding configuration file. Each router object will then verify which existent interfaces belong to it and create the necessary commands to configure them. After configuring the interfaces, the router object will then include the commands necessary to configure the routing protocol defined in the specification file. At the moment only RIPv2 and OSPF are supported. All these configurations are in the templates previously described in Section III. The configurations corresponding to each router are added to a plain text file, which is ready to be applied into a real or virtualized router. In the next section we describe the testing phase, used to assess the functionalities in a learning environment.

V. Tests and results analysis

Several tests were made in order to verify the correct functioning of the application. These tests consisted on designing a simple topology in GNS3 and using our application to generate the devices configuration files. As GNS3 is a very versatile tool, there are many options available to create a virtual network, as well as many different types of routers and switches and also the possibility to integrate those in a real scenario. We have used the version 1.2.3 of GNS3, which is the latest stable released version.

Our application has to read a specification file, which contains a variables set with paths to JSON file and to router baseline configurations, as well as the routing protocol to be used. As more features are included in our application, more variables would be added to the this specification file. An example of a specification file can be seen in figure 5.

In the working scenarios we have verified the correctness of the configuration files produced, as well as the performance achieved in the overall configuration procedure. In all the

```
ProjectDir = C:\Users\Script\scenario.gns3
PortForwarding = OSPF
BaseConfigFile =
C:\Users\Basefiles\baseConfig.txt
```

Figure 5. Example of the run specification file.

scenarios we have analysed the following functions: i) correct JSON file parsing and data storage; ii) correct configuration files produced automatically, with coherent IP configuration, by using the templates previously defined. In all the scenarios we applied the resulting configuration files both in physical network devices and virtual ones managed by GNS3. In both cases, each device was able to communicate with all the other devices in the network. Also the routing protocol was correctly configured and all the routers were able to learn the routes to reach the IP networks that were configured on the topology. In the next subsections we describe two of those tests, on trying to illustrate the appropriateness of using this application to automatically generate network devices configuration files.

A. *Scenario 1 - lab setup with two routers*

Figure 6 illustrates a basic testing scenario with two routers, a switch and two hosts. This scenario represents a simple topology which is used in the early Computer Networking lessons, where beginner practitioners take their first steps on configuring a small network. This setup was used in order to test the automatic generation of configuration files using the RIPv2 forwarding protocol, which was defined in the specification file.

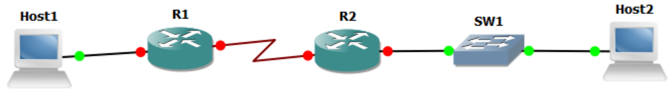


Figure 6. Simple two routers scenario.

Our application produced two configuration files for each one of the routers in the network topology. Three IP networks were identified in this topology and the routers configuration files reflected this fact. This means that there was a correct interpretation of the topology and interaction between the nodes. RIPv2 was also configured, meaning each router would announce to its neighbour all the networks it has configured. Figure 7 depicts an extract of the configuration file generated for router R1.

As can be observed, the interfaces have a valid IP configuration and the RIP configuration is correctly configured, by identifying the version 2 and by advertising the networks. After loading the resulting configuration files onto the routers, the network devices were able to communicate properly. That is, we were able to simply `ping` from `Host1` to `Host2`.

B. *Scenario 2 - lab setup with sixteen routers*

The next scenario, depicted in Figure 8, has sixteen routers interconnected by link-layer OSPF routing protocol. This

```
ip cef
no ip domain lookup
no ip icmp rate-limit unreachable
ip tcp synwait 5
no cdp log mismatch duplex
!
line con 0
 exec-timeout 0 0
 logging synchronous
 privilege level 15
 no login
line aux 0
 exec-timeout 0 0
 logging synchronous
 privilege level 15
 no login
!
interface FastEthernet0/0
ip address 192.168.1.2 255.255.255.0
no shutdown
!
interface Serial0/0
ip address 192.168.2.1 255.255.255.0
no shutdown
clock rate 2000000
!
router rip
version 2
network 192.168.1.0
network 192.168.2.0
!
end
```

Figure 7. Extract of the configuration file for Router R1.

scenario emulates a network with an access layer, a distribution layer and a backbone layer.

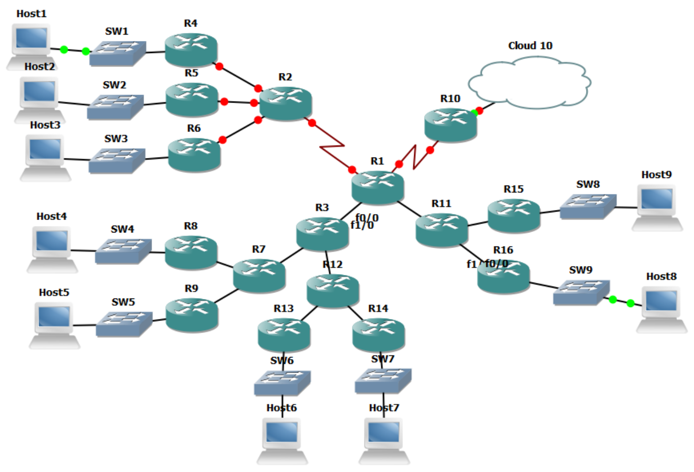


Figure 8. A scenario with sixteen interconnected routers.

After automatically configuring this scenario, we have obtained sixteen configuration files, each named with the router identification and containing the correct configurations. Figure 9 depicts the configuration file produced for router R1. In this lab setup we were able to evaluate the robustness of the template for OSPF routing protocol and the correctness of route announcements. After the automatic configuration procedure we were able to `ping` between Host1 and Host8.

The most time consuming operations are data processing and automatic generation of configuration files, tasks in which Perl excels. We have observed during the tests that Perl script's performance does not decay significantly when in the presence of too complex network topologies. The tests were deemed successful in proving the efficiency, utility and advantages of using the script in various network scenarios.

```
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname R1
!
ip cef
no ip domain lookup
no ip icmp rate-limit unreachable
ip tcp synwait 5
no cdp log mismatch duplex
!
line con 0
 exec-timeout 0 0
 logging synchronous
 privilege level 15
 no login
line aux 0
 exec-timeout 0 0
 logging synchronous
 privilege level 15
 no login
!
interface Serial0/0
ip address 192.168.4.2 255.255.255.0
no shutdown
clock rate 2000000
!
interface FastEthernet0/0
ip address 192.168.9.2 255.255.255.0
no shutdown
!
interface Serial0/1
ip address 192.168.12.1 255.255.255.0
no shutdown
clock rate 2000000
!
interface FastEthernet0/1
ip address 192.168.13.1 255.255.255.0
no shutdown
!
router ospf 1
network 192.168.4.0 0.0.0.255 area 0
network 192.168.9.0 0.0.0.255 area 0
network 192.168.12.0 0.0.0.255 area 0
network 192.168.13.0 0.0.0.255 area 0
!
end
```

Figure 9. Extract of the configuration file for Router R1.

VI. Conclusions and Future Work

In this paper we have proposed an extension, in the form of a set of Perl scripts, to GNS3 which automatically generates the Cisco IOS initial configuration files of all the network devices that are part of a topology. We have made tests with complex network topologies and observed the following two main advantages on using this application in computers networking classes. First, to setup small network lab scenarios are frequently in early learning environments, in which beginner users can take advantage of this solution for comparison with their own configurations. This helps the beginner practitioners to learn the most common configuration mistakes and troubleshoot their own configurations. Second, in more advanced topologies composed of dozens of routers, this solution can be approached differently. For example, in a scenario with 20 routers, where the main goal is to implement Multi-Protocol Label Switching (MPLS), advanced users need a functioning network to configure such protocol. Whereas these users already have the basic configuration notions well assimilated, they shouldn't be wasting time applying the initial configuration in the 20 routers, which is a repetitive and error-prone activity. This solution could have these users skip this step, providing all the necessary configurations as to configure the scenario and prepare it for the MPLS configuration. GNS3 would the benefit with the integration of a solution for automatic configuration, regardless of the size of the scenario being tested.

Our application takes advantage of the GNS3 specification files generated during the design of the network topology. It also uses a run configuration file with specific settings used during the processing. Combining GNS3 with this extension gives an easier and faster way to automatically configure error-prone network that are ready to be tested in a lab environment.

We are now working on making available more specific network configurations, such as VLANs, ACLs and OSPF with multi-area and other routing protocols. Another development that is taking place consists on implementing an automatic configuration generator for a multi-vendor network topology, that may include not only Cisco IOS equipments, but also Huawei and Alcatel-Lucent. Both implementations rely on the development of templates for each one of the previously described features.

We believe to have fulfilled successfully the initial goals and to have proven the robustness of our solution in networking lab setups, giving a learning environment for beginners and more advanced students and practitioners the ability to better learn networking topics in classroom and in self-study.

VII. Acknowledgements

This work is financed by the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project UID/EEA/50014/2013.

References

[1] A. Nogueira and P. Salvador, "Teaching networking: A hands-on approach that relies on emulation-based projects," in *INFOCOMP 2014, The Fourth International Conference on Advanced Communications and Computation*, 2014, pp. 149–155.

[2] I. ACM, "Computer science curricula 2013 - curriculum guidelines for undergraduate degree programs in computer science," 2013.

[3] L. Sun, Y. Zhang, H. Yin *et al.*, "Comparison between physical devices and simulator software for cisco network technology teaching," in *Computer Science & Education (ICCSE), 2013 8th International Conference on*. IEEE, 2013, pp. 1357–1360.

[4] A. Wang, M. Iyer, R. Dutta, G. N. Rouskas, and I. Baldine, "Network virtualization: Technologies, perspectives, and frontiers," *Lightwave Technology, Journal of*, vol. 31, no. 4, pp. 523–537, 2013.

[5] S. Siraj, A. Gupta, and R. Badgujar, "Network simulation tools survey," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, no. 4, pp. 199–206, 2012.

[6] A. Jesin, *Packet Tracer Network Simulator*. Packt Publishing Ltd, 2014.

[7] W.-J. Hsin, "Learning computer networking through illustration," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 2015, pp. 515–515.

[8] C. S. Tan, "Network simulator test engine for huawei ensp and cisco gns3," Ph.D. dissertation, UTAR, 2014.

[9] Y. Liu, "The application of gns3 in network equipment of the internet course teaching," *Computer Knowledge and Technology*, vol. 8, p. 057, 2012.

[10] C. Welsh, *GNS3 network simulation guide*. Packt Publ., 2013.

[11] S. Knight, A. Jaboldinov, O. Maennel, I. Phillips, and M. Roughan, "Autonetkit: simplifying large scale, open-source network experimentation," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 2012, pp. 97–98.